

**Oracle® Application Server Web Services**

Developer's Guide

10g Release 2 (10.1.2)

**Part No. B14027-01**

November 2004

Oracle Application Server Web Services Developer's Guide, 10g Release 2 (10.1.2)

Part No. B14027-01

Copyright © 2001, 2004 Oracle. All rights reserved.

Primary Author: Thomas Van Raalte

Contributing Author: Rodney Ward

Contributors: Jeremy Blanchard, Marco Carrer, Anirban Chatterjee, Daxin Cheng, David Clay, Tony D'Silva, Neil Evans, Bert Feldman, Kathryn Gruenefeldt, Steven Harris, Anish Karmarkar, Prabha Krishna, Sunil Kunisetty, Wai-Kwong (Sam) Lee, Gary Moyer, Steve Muench, Giuseppe Panciera, Wei Qian, Eric Rajkovic, Venkata Ravipati, Susan Shepard, Alok Srivastava, Rodney Ward, Zhe (Alan) Wu, Joyce Yang, Chen Zhou

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

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

If the Programs are delivered to the United States Government or anyone licensing or using the Programs on behalf of the United States Government, the following notice is applicable:

U.S. GOVERNMENT RIGHTS Programs, software, databases, and related documentation and technical data delivered to U.S. Government customers are "commercial computer software" or "commercial technical data" pursuant to the applicable Federal Acquisition Regulation and agency-specific supplemental regulations. As such, use, duplication, disclosure, modification, and adaptation of the Programs, including documentation and technical data, shall be subject to the licensing restrictions set forth in the applicable Oracle license agreement, and, to the extent applicable, the additional rights set forth in FAR 52.227-19, Commercial Computer Software--Restricted Rights (June 1987). Oracle Corporation, 500 Oracle Parkway, Redwood City, CA 94065

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

Oracle is a registered trademark of Oracle Corporation and/or its affiliates. Other names may be trademarks of their respective owners.

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

---

---

# Contents

<b>Send Us Your Comments</b> .....	xi
<b>Preface</b> .....	xiii
Intended Audience .....	xiv
Documentation Accessibility .....	xiv
Organization .....	xiv
Related Documentation .....	xvi
Conventions .....	xvi
<b>1 Web Services Overview</b>	
<b>What Are Web Services?</b> .....	1-1
Understanding Web Services .....	1-1
Benefits of Web Services .....	1-2
About the Web Services e-Business Transformation .....	1-2
<b>Overview of Web Services Standards</b> .....	1-3
SOAP Standard .....	1-4
Web Services Description Language (WSDL) .....	1-5
Universal Description, Discovery, and Integration (UDDI) .....	1-5
<b>SOAP Message Exchange and SOAP Message Encoding</b> .....	1-5
SOAP Message Components .....	1-5
Working With RPC Style SOAP Messages .....	1-6
Working With Document Style SOAP Messages .....	1-6
<b>2 Oracle Application Server Web Services</b>	
<b>Oracle Application Server OC4J (J2EE) and Oracle SOAP Based Web Services</b> .....	2-1
<b>Oracle Application Server Web Services Standards</b> .....	2-2
<b>Oracle Application Server Web Services Features</b> .....	2-2
Developing End-to-End Web Services .....	2-2
Deploying and Managing Web Services .....	2-3
Using Oracle JDeveloper with Web Services .....	2-3
Securing Web Services .....	2-4
Aggregating Web Services .....	2-4
<b>Oracle Application Server Web Services Architecture</b> .....	2-4
About Servlet Entry Points for Web Services .....	2-5
What Are the Packaging and Deployment Options for Web Services .....	2-7

About Server Skeleton Code Generation for Web Services .....	2-7
<b>Understanding WSDL and Client Proxy Stubs for Web Services .....</b>	<b>2-8</b>
Overview of a WSDL Based Web Service Client.....	2-9
Overview of a Client-Side Proxy Stubs Based Web Service Client.....	2-9
<b>Web Services Home Page.....</b>	<b>2-9</b>
<b>About Universal Description, Discovery, and Integration Registry .....</b>	<b>2-10</b>
Oracle Enterprise Manager Features to Register Web Services .....	2-11
<b>3 Developing and Deploying Java Class Web Services</b>	
<b>Using Oracle Application Server Web Services With Java Classes .....</b>	<b>3-1</b>
<b>Writing Java Class Based Web Services .....</b>	<b>3-1</b>
Writing Stateless and Stateful Java Web Services .....	3-2
Building a Sample Java Class Implementation.....	3-2
Using Supported Data Types for Java Web Services .....	3-5
<b>Preparing and Deploying Java Class Based Web Services.....</b>	<b>3-6</b>
Creating a Configuration File to Assemble Java Class Web Services.....	3-6
Running WebServicesAssembler To Prepare Java Class Web Services.....	3-10
Deploying Java Class Based Web Services.....	3-11
<b>Serializing and Encoding Parameters and Results for Web Services .....</b>	<b>3-11</b>
<b>4 Developing and Deploying EJB Web Services</b>	
<b>Using Oracle Application Server Web Services With Stateless Session EJBs.....</b>	<b>4-1</b>
<b>Writing Stateless Session EJB Web Services .....</b>	<b>4-1</b>
Defining a Stateless Session Remote Interface .....	4-2
Defining a Stateless Session Home Interface .....	4-2
Defining a Stateless Session EJB Bean.....	4-3
Returning Results From EJB Web Services.....	4-4
Error Handling for EJB Web Services .....	4-4
Serializing and Encoding Parameters and Results for EJB Web Services.....	4-4
Using Supported Data Types for Stateless Session EJB Web Services .....	4-4
Writing a WSDL File for EJB Web Services (Optional).....	4-6
<b>Preparing and Deploying Stateless Session EJB Based Web Services.....</b>	<b>4-6</b>
Creating a Configuration File to Assemble Stateless Session EJB Web Services .....	4-6
Running WebServicesAssembler To Prepare Stateless Session EJB Web Services .....	4-9
Deploying Web Services Implemented as EJBs.....	4-9
<b>5 Developing and Deploying Stored Procedure Web Services</b>	
<b>Using Oracle Application Server Web Services with Stored Procedures.....</b>	<b>5-1</b>
<b>Writing Stored Procedure Web Services .....</b>	<b>5-2</b>
<b>Preparing Stored Procedure Web Services .....</b>	<b>5-2</b>
Creating a Configuration File to Assemble Stored Procedure Web Services.....	5-3
Running WebServicesAssembler With Stored Procedure Web Services.....	5-8
Setting Up Datasources in Oracle Application Server Web Services (OC4J) .....	5-8
<b>Deploying Stored Procedure Web Services.....</b>	<b>5-9</b>
<b>Limitations for Stored Procedures Running as Web Services .....</b>	<b>5-9</b>
Supported Stored Procedure Features for Web Services .....	5-9

	Unsupported Stored Procedure Features for Web Services .....	5-10
	Database Server Release Limitation for Boolean Use in Oracle PL/SQL Web Services .....	5-10
	TIMESTAMP and DATE Granularity Limitation.....	5-11
	LOB (CLOB/BLOB) Emulated Data Source Limitation .....	5-11
<b>6</b>	<b>Developing and Deploying Document Style Web Services</b>	
	<b>Using Document Style Web Services</b> .....	6-1
	<b>Writing Document Style Web Services</b> .....	6-2
	Supported Method Signatures for Document Style Web Services .....	6-2
	Writing Stateless and Stateful Document Style Web Services.....	6-3
	Writing Classes and Interfaces for Document Style Web Services .....	6-3
	<b>Preparing Document Style Web Services</b> .....	6-6
	Creating a Configuration File to Assemble Document Style Web Services .....	6-7
	Running WebServicesAssembler With Document Style Web Services .....	6-11
	<b>Deploying Document Style Web Services</b> .....	6-11
<b>7</b>	<b>Developing and Deploying JMS Web Services</b>	
	<b>JMS Web Services Overview</b> .....	7-1
	Using JMS Web Services .....	7-1
	JMS Web Services Backend Message Processing.....	7-2
	<b>Writing JMS Web Services and Handling Messages</b> .....	7-4
	Using an MDB for Backend Message Processing .....	7-5
	Using a JMS Standalone Program for Backend Message Processing .....	7-7
	Message Processing and Reply Messages .....	7-7
	<b>Preparing and Configuring JMS Web Services</b> .....	7-8
	Creating a Configuration File to Assemble JMS Web Services .....	7-9
	Running WebServicesAssembler With JMS Web Services .....	7-13
	<b>Deploying JMS Web Services</b> .....	7-13
	<b>Limitations for JMS Web Services</b> .....	7-14
<b>8</b>	<b>Building Clients that Use Web Services</b>	
	<b>Locating Web Services</b> .....	8-1
	<b>Getting WSDL Files and Client-Side Proxy Jars for Web Services</b> .....	8-2
	Using the Web Service Home Page to Save WSDL and Client Side Proxies.....	8-2
	Getting Web Service WSDL and Client-Side Proxies Directly .....	8-3
	Generating Client-Side Proxies With WebServicesAssembler .....	8-5
	<b>Working with Client-Side Proxy Jar to Use Web Services</b> .....	8-7
	Setting the Web Services Proxy Client CLASSPATH .....	8-8
	Using Java Beans as Parameters for Web Services .....	8-8
	Using Web Services Security Features .....	8-9
	<b>Working with WSDL Files and Oracle JDeveloper to Use Web Services</b> .....	8-10
<b>9</b>	<b>Web Services Tools</b>	
	<b>Running the Web Services Assembly Tool</b> .....	9-1
	<b>Web Services Assembly Tool Configuration File Sample</b> .....	9-1

Web Services Assembly Tool Configuration File Sample Output .....	9-2
<b>Generating WSDL Files and Client Side Proxies</b> .....	9-3
Generating and Assembling WSDL Files .....	9-3
Generating Client-Side Proxies with WSDL .....	9-5
<b>Web Services Assembly Tool Configuration File Specification</b> .....	9-6
<b>Web Services Assembly Tool Limitations</b> .....	9-8

## 10 Discovering and Publishing Web Services

<b>Understanding a UDDI Registry</b> .....	10-1
UDDI Registry Data Structure .....	10-2
<b>Introducing OracleAS UDDI Registry</b> .....	10-3
Support for Standard Classification and Identifier Systems .....	10-4
UUID Generation .....	10-5
<b>Getting Started with OracleAS UDDI Registry</b> .....	10-5
Configuring OracleAS UDDI Registry.....	10-6
Modifying Properties at Installation or First-Use.....	10-7
Considerations in a Production Environment .....	10-8
<b>Web Services Discovery</b> .....	10-9
Using the OracleAS UDDI Registry Searching and Browsing Tool .....	10-9
Using Other Tools to Discover Web Services.....	10-9
Using the OracleAS UDDI Registry Inquiry API .....	10-10
<b>Web Services Publishing</b> .....	10-13
Using Oracle Enterprise Manager for Web Services Publishing.....	10-13
Using the OracleAS UDDI Registry Publishing Tool .....	10-19
Using the OracleAS UDDI Registry Publishing API .....	10-28
<b>OracleAS UDDI Registry Administration</b> .....	10-34
Using the Command-Line Tool uddiadmin.jar .....	10-34
Configuring the Server .....	10-35
Managing Users.....	10-36
Enforcing Quotas.....	10-37
Managing Administrative Entities .....	10-39
Importing Entities .....	10-40
Setting Operational Information.....	10-41
UDDI Replication.....	10-41
Registry-Based Category Validation .....	10-44
External Validation .....	10-47
Performance Monitoring and Tuning.....	10-48
Data Backup and Restore Operations .....	10-49
Database Configuration .....	10-49
Transport Security.....	10-50
<b>UDDI Open Database Support</b> .....	10-50
Microsoft SQL Server.....	10-51
IBM DB2.....	10-53
Other Oracle Database (Non-Infrastructure) .....	10-55
<b>OracleAS UDDI Registry Server Error Messages</b> .....	10-57

## 11 Consuming Web Services in J2EE Applications

<b>Consuming SOAP-Based Web Services Using WSDL</b> .....	11-1
Advanced Configuration .....	11-3
Known Limitations of the <code>wsdl2ejb</code> Utility.....	11-7
Running the Demonstration .....	11-7
<b>Dynamic Invocation of Web Services</b> .....	11-16
Dynamic Invocation API.....	11-16
WebServiceProxy Client.....	11-19
Known Limitations .....	11-21

## 12 Advanced Topics for Web Services

<b>Setting the Web Services Debugging Property <code>ws.debug</code></b> .....	12-1
<b>Untyped Request Handling Options</b> .....	12-1
<b>SOAP Header Support</b> .....	12-3
Client Side SOAP Request Header Support.....	12-3
Server Side SOAP Request Header Support .....	12-4
Limitations for SOAP Header Support .....	12-6

## A Using Oracle Application Server SOAP

<b>Understanding Oracle Application Server SOAP</b> .....	A-2
<b>Apache SOAP Documentation</b> .....	A-2
<b>Configuring the SOAP Request Handler Servlet</b> .....	A-2
<b>Using OracleAS SOAP Management Utilities and Scripts</b> .....	A-4
Managing Providers .....	A-5
Using the Service Manager to Deploy and Undeploy Java Services .....	A-5
Generating Client Proxies from WSDL Documents.....	A-6
Generating WSDL Documents from Java Service Implementations .....	A-7
<b>Deploying OracleAS SOAP Services</b> .....	A-7
Creating Deployment Descriptors.....	A-8
Installing a SOAP Web Service in OC4J.....	A-9
Disabling an Installed SOAP Web Service .....	A-9
Installing a SOAP Web Service in an OC4J Cluster .....	A-9
<b>Using OracleAS SOAP Handlers</b> .....	A-10
Request Handlers .....	A-10
Response Handlers .....	A-10
Error Handlers.....	A-10
Configuring Handlers .....	A-10
<b>Using OracleAS SOAP Audit Logging</b> .....	A-11
Audit Logging Information .....	A-11
Auditable Events .....	A-12
Configuring the Audit Logger .....	A-13
<b>Using OracleAS SOAP Pluggable Configuration Managers</b> .....	A-15
<b>Working With OracleAS SOAP Transport Security</b> .....	A-15
Apache Listener and Servlet Engine Configuration for SSL.....	A-18
Using JSSE with Oracle Application Server SOAP Client.....	A-18
<b>Using OracleAS SOAP Sample Services</b> .....	A-20

The Xmethods Sample.....	A-20
The AddressBook Sample.....	A-20
The StockQuote Sample.....	A-20
The Company Sample.....	A-20
The Provider Sample.....	A-21
The AddressBook2 Sample.....	A-21
The Messaging Sample.....	A-21
The Mime Sample.....	A-21
<b>Using the OracleAS SOAP EJB Provider.....</b>	<b>A-21</b>
Stateless Session EJB Provider.....	A-21
Stateful Session EJB Provider in Apache SOAP.....	A-21
Stateful Session EJB Provider in OracleAS SOAP.....	A-22
Entity EJB Provider in OracleAS SOAP.....	A-22
Deployment and Use of the OracleAS SOAP EJB Provider.....	A-22
Current Known EJB Provider Limitations.....	A-23
<b>Using PL/SQL Stored Procedures With the SP Provider.....</b>	<b>A-23</b>
SP Provider Supported Functionality.....	A-23
SP Provider Unsupported Functionality.....	A-23
SP Provider Supported Simple PL/SQL Types.....	A-24
Using Object Types.....	A-24
Deploying a Stored Procedure Provider.....	A-24
Translating PL/SQL Stored Procedures into Java.....	A-24
Deploying a Stored Procedure Service.....	A-25
Invoking a SOAP Service that is a Stored Procedure.....	A-26
<b>SOAP Troubleshooting and Limitations.....</b>	<b>A-26</b>
Tunneling Using the TcpTunnelGui Command.....	A-26
Setting Configuration Options for Debugging.....	A-27
Using DMS to Display Runtime Information.....	A-27
SOAP Limitations for Java Type Precedence with Overloaded Methods.....	A-27
<b>OracleAS SOAP Differences From Apache SOAP.....</b>	<b>A-28</b>
Service Installation Differences.....	A-28
Optional Provider Enhancements.....	A-28
Oracle Transport libraries.....	A-28
Modifications to Apache EJB Provider.....	A-28
Stored Procedure Provider.....	A-29
Utility Enhancements.....	A-29
Modifications to Sample Code.....	A-29
Handling the mustUnderstand Attribute in the SOAP Header.....	A-29
<b>Apache Software License, Version 1.1.....</b>	<b>A-30</b>
<b>B Web Services Security</b>	
<b>About Web Services Security.....</b>	<b>B-2</b>
<b>Configuring Web Services Security.....</b>	<b>B-2</b>
<b>About Oracle Application Server UDDI Registry Security.....</b>	<b>B-4</b>
Protecting Oracle Application Server UDDI Registry Resources.....	B-4
Managing and Enforcing Protected UDDI Resources.....	B-5
Using Oracle Application Server Security Services.....	B-5



<b>Configuring UDDI Security</b> .....	B-5
Configuring the Oracle Application Server UDDI Registry .....	B-5
Configuring the UDDI Content Subscription Manager .....	B-6
Configuring the UDDI Client.....	B-6

## **C Troubleshooting OracleAS Web Services**

<b>Problems and Solutions</b> .....	C-1
Receiving "Unsupported Response Content Type" Error .....	C-1
Cannot Publish Doc/Literal in JDeveloper .....	C-1
Cannot Register Web Service .....	C-2
UDDI Registry Screens Missing.....	C-2
UDDI Management Screens Are Not Enabled .....	C-2
<b>Diagnosing OracleAS Web Services Problems</b> .....	C-3
Generating Web Services Diagnostic Messages .....	C-3
<b>Need More Help?</b> .....	C-3

## **Glossary**

## **Index**



---

---

# Send Us Your Comments

**Oracle Application Server Web Services Developer's Guide, 10g Release 2  
(10.1.2)**

**Part No. B14027-01**

Oracle welcomes your comments and suggestions on the quality and usefulness of this publication. Your input is an important part of the information used for revision.

- Did you find any errors?
- Is the information clearly presented?
- Do you need more information? If so, where?
- Are the examples correct? Do you need more examples?
- What features did you like most about this manual?

If you find any errors or have any other suggestions for improvement, please indicate the title and part number of the documentation and the chapter, section, and page number (if available). You can send comments to us in the following ways:

- Electronic mail: [appserverdocs\\_us@oracle.com](mailto:appserverdocs_us@oracle.com)
- FAX: 650-506-7375 Attn: Oracle Application Server Documentation Manager
- Postal service:

Oracle Corporation  
Oracle Application Server Documentation  
500 Oracle Parkway M/S 10p6  
Redwood Shores, CA 94065  
USA

If you would like a reply, please give your name, address, telephone number, and electronic mail address (optional).

If you have problems with the software, please contact your local Oracle Support Services.



---

---

# Preface

This guide describes Oracle Application Server Web Services.

This preface contains these topics:

- [Intended Audience](#)
- [Documentation Accessibility](#)
- [Organization](#)
- [Related Documentation](#)
- [Conventions](#)

## Intended Audience

*Oracle Application Server Web Services Developer's Guide* is intended for application programmers, system administrators, and other users who perform the following tasks:

- Configure software installed on the Oracle Application Server.
- Create programs that implement Web Services
- Create programs that run as Web Services clients

To use this document, you need a working knowledge of Java programming language fundamentals.

## Documentation Accessibility

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

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

**Accessibility of Code Examples in Documentation** JAWS, a Windows screen reader, may not always correctly read the code examples in this document. The conventions for writing code require that closing braces should appear on an otherwise empty line; however, JAWS may not always read a line of text that consists solely of a bracket or brace.

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

## Organization

This document contains:

### **Chapter 1, "Web Services Overview"**

This chapter provides an overview of Oracle Application Server Web Services.

### **Chapter 2, "Oracle Application Server Web Services"**

This chapter describes the Oracle Application Server Web Services features, architecture, and implementation.

### **Chapter 3, "Developing and Deploying Java Class Web Services"**

This chapter describes the procedures you use to write and deploy Oracle Application Server Web Services that are implemented as Java classes.

#### **Chapter 4, "Developing and Deploying EJB Web Services"**

This chapter describes the procedures you use to write and deploy Oracle Application Server Web Services that are implemented as stateless session Enterprise Java Beans (EJBs).

#### **Chapter 5, "Developing and Deploying Stored Procedure Web Services"**

This chapter describes the procedures you use to write and deploy Oracle Application Server Web Services that are implemented as PL/SQL Stored Procedures or Functions.

#### **Chapter 6, "Developing and Deploying Document Style Web Services"**

This chapter describes the procedures you use to write and deploy Document Style Oracle Application Server Web Services implemented as Java classes.

#### **Chapter 7, "Developing and Deploying JMS Web Services"**

This chapter describes the procedures you use to write and deploy Oracle Application Server Web Services that expose JMS destinations as Web Services.

#### **Chapter 8, "Building Clients that Use Web Services"**

This chapter describes the steps required to build a client application that uses Oracle Application Server Web Services.

#### **Chapter 9, "Web Services Tools"**

This chapter describes the Oracle Application Server Web Services assembly tool, `WebServicesAssembler`, that assists in assembling Oracle Application Server Web Services.

#### **Chapter 10, "Discovering and Publishing Web Services"**

This chapter provides a description of the Universal Discovery Description and Integration (UDDI-compliant Web Services registry in which business Web Service providers in an enterprise environment can publish and describe their Web Services.

#### **Chapter 11, "Consuming Web Services in J2EE Applications"**

This chapter describes ways to consume Web Services in J2EE applications.

#### **Chapter 12, "Advanced Topics for Web Services"**

This chapter describes several advanced Oracle Application Server Web Services topics, including untyped request handling options and SOAP header support.

#### **Appendix A, "Using Oracle Application Server SOAP"**

This appendix describes Oracle SOAP and covers the differences between Apache SOAP and Oracle SOAP.

#### **Appendix B, "Web Services Security"**

This appendix describes the architecture and configuration of security for Oracle Application Server Web Services, including the Oracle Application Server UDDI Registry.

#### **Appendix C, "Troubleshooting OracleAS Web Services"**

This appendix provides information on troubleshooting problems with Web services.

## Glossary

The glossary contains the Web Services glossary terms and descriptions.

## Related Documentation

For more information, see these Oracle resources:

- Overview Guide in the Oracle Application Server 10g Documentation Library.
- Oracle Application Server Containers for J2EE User's Guide in the Oracle Application Server 10g Documentation Library.

Printed documentation is available for sale in the Oracle Store at

<http://oraclestore.oracle.com/>

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

<http://otn.oracle.com/membership/>

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

<http://otn.oracle.com/documentation/content.html>

## Conventions

The following conventions are used in this manual:

Convention	Meaning
...	Ellipsis points in statements or commands mean that parts of the statement or command not directly related to the example have been omitted
<b>boldface text</b>	Boldface type in text indicates a term defined in the text, the glossary, or in both locations.
[ ]	Brackets enclose optional clauses from which you can choose one or none.
\$	The dollar sign represents the Command Language prompt in Windows and the Bourne shell prompt in UNIX



---

---

# Web Services Overview

This chapter provides an overview of Web Services. [Chapter 2, "Oracle Application Server Web Services"](#) describes the Oracle Application Server Web Services features, architecture, and implementation.

This chapter covers the following topics:

- [What Are Web Services?](#)
- [Overview of Web Services Standards](#)
- [SOAP Message Exchange and SOAP Message Encoding](#)

## What Are Web Services?

Web Services consist of a set of messaging protocols, programming standards, and network registration and discovery facilities that expose business functions to authorized parties over the Internet from any web-connected device.

This section covers the following topics:

- [Understanding Web Services](#)
- [Benefits of Web Services](#)
- [About the Web Services e-Business Transformation](#)

## Understanding Web Services

A **Web Service** is a software application identified by a URI, whose interfaces and binding are capable of being defined, described, and discovered by XML artifacts. A Web Service supports direct interactions with other software applications using XML based messages and internet-based products.

A Web Service does the following:

- Exposes and describes itself – A Web Service defines its functionality and attributes so that other applications can understand it. By providing a WSDL file, a Web Service makes its functionality available to other applications.
- Allows other services to locate it on the web – A Web Service can be registered in a UDDI Registry so that applications can locate it.
- Can be invoked – Once a Web Service has been located and examined, the remote application can invoke the service using an Internet standard protocol.
- Web Services are of either request and response or one-way style, and they can use either synchronous or asynchronous communication. However, the fundamental

unit of exchange between Web Services clients and Web Services, of either style or type of communication, is a message.

Web Services provide a standards based infrastructure through which any business can do the following:

- Offer appropriate internal business processes as value-added services that can be used by other organizations.
- Integrate its internal business processes and dynamically link them with those of its business partners.

## Benefits of Web Services

The benefits for enterprises seeking to develop and use Web Services to streamline their business processes include the following:

- Support for open Internet standards. Oracle supports SOAP, WSDL, and UDDI as the primary standards to develop Web Services. Web Services developed with Oracle's products can inter-operate with those developed to Microsoft's .NET architecture.
- Simple and productive development facilities. Oracle provides developers with an easy-to-use and productive environment for developing Web Services using a programming model that is identical to that for J2EE applications.
- Mission critical deployment facilities. Oracle provides a mission-critical platform to deploy Web Services by unifying the Web Services and J2EE runtime infrastructure. Oracle Application Server Web Services provide optimizations to speed up Web Services responses, to scale Web Services on single CPUs or multiple CPUs, and to provide high availability through fault tolerant design and clustering.

**See Also:** ["Overview of Web Services Standards"](#) on page 1-3

## About the Web Services e-Business Transformation

The move to transform businesses to e-Businesses has driven organizations around the world to begin to use the Internet to manage corporate business processes. Despite this transformation, business on the Internet still functions as a set of local nodes, or Web sites, with point-to-point communications between them. As more business moves online, the Internet should no longer be used in such a static manner, but rather should be used as a universal business network through which services can flow freely, and over which applications can interact and negotiate among themselves.

To enable this transformation, the Internet needs to support a standards-based infrastructure that enables companies and their enterprise applications to communicate with other companies and their applications more efficiently. These standards should allow discrete business processes to expose and describe themselves on the Internet, allow other services to locate them, to invoke them once they have been located, and to provide a predictable response.

Web Services drive this transformation by promising a fundamental change in the way businesses function and enterprise applications are developed and deployed.

This e-Business transformation is occurring in the following two areas:

- Business Transformation with Web Services
- Technology Transformation with Web Services

### **About Business Transformation with Web Services**

Web Services enables the next-generation of e-business, a customer-centric, agile enterprise that does the following:

- Expands Markets - Offers business processes to existing and new customers as services over the Internet, opening new global channels and capturing new revenue opportunities.
- Improves Efficiencies - Streamlines business processes across the entire enterprise and with business partners, taking action in real-time with up-to-date information.
- Reaches Suppliers and Partners - Creates and maintains pre-defined, systematic, contractually negotiated relationships and dynamic, spot partnerships with business partners who are tightly linked within supply chains.

### **About Technology Transformation with Web Services**

Web Services enables enterprise applications with the following technology transformations:

- Development and Deployment – Web Services can be developed and deployed quickly and productively.
- Locating Services – Web Services allow applications to be aggregated and discovered within Internet portals, enterprise portals, or service registries which serve as Internet *Yellow Pages*.
- Integrating Services – Web Services allow applications to locate and electronically communicate with other applications within an enterprise and outside the enterprise boundaries.
- Inter-Operating Services – Web Services allow applications to inter-operate with applications that are developed using different programming languages and following different component paradigms.

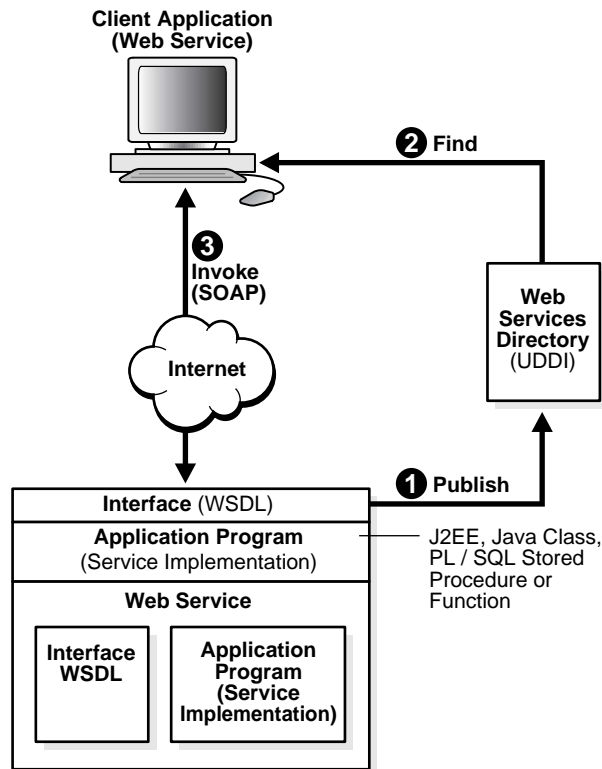
## **Overview of Web Services Standards**

This section describes the Internet standards that comprise Web Services, including:

- [SOAP Standard](#)
- [Web Services Description Language \(WSDL\)](#)
- [Universal Description, Discovery, and Integration \(UDDI\)](#)

[Figure 1-1](#) shows a conceptual architecture for Web Services using these standards.

Figure 1–1 Web Services Standards



## SOAP Standard

The **SOAP** is a lightweight, XML-based protocol for exchanging information in a decentralized, distributed environment. SOAP supports different styles of information exchange, including: Remote Procedure Call style (RPC) and Message-oriented exchange. **RPC style** information exchange allows for request-response processing, where an endpoint receives a procedure oriented message and replies with a correlated response message. **Message-oriented** information exchange supports organizations and applications that need to exchange business or other types of documents where a message is sent but the sender may not expect or wait for an immediate response. Message-oriented information exchange is also called **Document style** exchange.

SOAP has the following features:

- Protocol independence
- Language independence
- Platform and operating system independence
- Support for SOAP XML messages incorporating attachments (using the multipart MIME structure)

**See Also:** <http://www.w3.org/TR/SOAP/> for information on the SOAP 1.1 specification

## Web Services Description Language (WSDL)

The Web Services Description Language (WSDL) is an XML format for describing network services containing RPC-oriented and message-oriented information. Programmers or automated development tools can create WSDL files to describe a service and can make the description available over the Internet. Client-side programmers and development tools can use published WSDL descriptions to obtain information about available Web Services and to build and create proxies or program templates that access available services.

**See Also:** <http://www.w3.org/TR/wsdl> for information on the Web Services Description Language (WSDL) format.

## Universal Description, Discovery, and Integration (UDDI)

The Universal Description, Discovery, and Integration (UDDI) specification is an online electronic registry that serves as electronic *Yellow Pages*, providing an information structure where various business entities register themselves and the services they offer through their WSDL definitions.

There are two types of UDDI registries, public UDDI registries that serve as aggregation points for a variety of businesses to publish their services, and private UDDI registries that serve a similar role within organizations.

**See Also:** <http://www.uddi.org> for information on Universal Description, Discovery and Integration specifications.

## SOAP Message Exchange and SOAP Message Encoding

The SOAP standard defines a lightweight, XML-based protocol for exchanging information in a decentralized, distributed environment. SOAP supports different styles of information exchange, including: Remote Procedure Call, **RPC Style**, and Message-oriented exchange, or **Document Style**. SOAP Messages, whether RPC Style or Document Style use a certain encoding, as specified with the `encodingStyle` attribute specified for SOAP message elements. This section describes these SOAP message features, in the following sections:

- [SOAP Message Components](#)
- [Working With RPC Style SOAP Messages](#)
- [Working With Document Style SOAP Messages](#)

## SOAP Message Components

Each SOAP message is a transmission between a SOAP sender and a SOAP receiver. Each SOAP message consists of a SOAP envelope containing two sub-elements, a `Header` and a `Body`. The SOAP Header is optional. The children of the SOAP header are called `header blocks`; each header block represents a logical grouping of data. The SOAP Body is a mandatory element within a SOAP message. This is where the end-to-end information conveyed in a SOAP message is carried. The choice of what data is placed in a header block and what data goes in the SOAP Body element are decisions that are taken at the time that an application is designed.

Using Oracle Application Server Web Services, developers determine if an implementation supports RPC Style or Document Style messages. Developers write the appropriate application logic and the `WebServicesAssembler` configuration files for the implementation.

## Working With RPC Style SOAP Messages

Oracle Application Server Web Services supports two types of SOAP message exchanges: RPC Style exchanges and Document-Style exchanges. RPC Style exchanges represent exchanges that can be modeled as remote procedure calls (RPC); these are used when there is a need to model a certain programmatic behavior, with the exchanged messages conforming to a well-defined signature for the remote call and its return. Using RPC Style messages, SOAP specifies the form of the SOAP message body.

RPC style information exchange allows for request-response processing, where an endpoint receives a procedure oriented message and replies with a response message. Using the RPC style SOAP message exchange, the contents of the SOAP message body conform to a structure that specifies a procedure and includes set of parameters, or a response, with a result and any additional parameters. The SOAP message in the body is an XML document, but it is XML document that conforms the limitations specified in the SOAP specification.

[Example 1-1](#) shows a SOAP RPC Style request that includes the `ChargeReservation` method with several parameters. [Example 1-2](#) shows the SOAP RPC Style response message that includes the `ChargeReservationResponse`, with a "Response" string appended.

### **Example 1-1 SOAP RPC Style Request Message**

```
<?xml version='1.0' encoding='UTF-8'?>
<SOAP-ENV:Envelope xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <SOAP-ENV:Body>
    <ns1:helloWorld xmlns:ns1="urn:oracle-j2ee-ws_example-StatelessExample"
      SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
      <param0 xsi:type="xsd:string">Wendy</param0>
    </ns1:helloWorld>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

### **Example 1-2 SOAP RPC Style Response Message**

```
<?xml version='1.0' encoding='UTF-8'?>
<SOAP-ENV:Envelope xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <SOAP-ENV:Body>
    <ns1:helloWorldResponse
      xmlns:ns1="urn:oracle-j2ee-ws_example-StatelessExample"
      SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
      <return xsi:type="xsd:string">Hello World, Wendy</return>
    </ns1:helloWorldResponse>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

## Working With Document Style SOAP Messages

Oracle Application Server Web Services supports two types of SOAP message exchanges: RPC Style exchanges and Document-Style exchanges. Document-style exchanges, also called message-oriented exchanges, model exchanges where XML

documents are exchanged, where the exchange patterns are defined in the sending and the receiving applications. For Document Style messages, SOAP places no constraints on how the document sent in the SOAP message body is structured, the application, or an externally specified XML schema determines the structure of the XML document that is sent in the body of the SOAP message.

Message-oriented information exchange supports organizations and applications that need to exchange business or other types of documents where a message is sent but the sender may not expect or wait for an immediate response. Message-oriented information exchange is also called Document style SOAP message exchange. Document -style messages model exchanges where XML documents are exchanged, where the semantics of the exchange patterns are defined in the sending and the receiving applications.

**Example 1-3** shows a sample Document Style SOAP message that is sent from a client to an Oracle Application Server Web Services document style service. The client sends an XML document that contains employee records with elements including name, emp\_id, department, and contact information. A web service that processes this XML document to produce a phone listing may supply an XML document that contains only the name and phone number elements.

### **Example 1-3 Document Style SOAP Message**

```
<?xml version='1.0' encoding='UTF-8'?>
<SOAP-ENV:Envelope xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:xsd="http://www.w3.org/2001/XMLSchema">
<SOAP-ENV:Body>
<organisation>
  <employee>
    <name>Bob</name>
    <emp_id>1234</emp_id>
    <department>hr</department>
    <contact>
      <phone>827 644 5674</phone>
      <email>bob@organisation.com</email>
    </contact>
  </employee>
  <employee>
    <name>Susan</name>
    <emp_id>2434</emp_id>
    <department>it</department>
    <contact>
      <phone>827 644 5674</phone>
      <email>Susan@organisation.com</email>
    </contact>
  </employee>
</organisation>
</SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

### **Example 1-4 Document Style SOAP Message Processed by a Web Service**

```
<?xml version='1.0' encoding='UTF-8'?>
<SOAP-ENV:Envelope xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:xsd="http://www.w3.org/2001/XMLSchema">
<SOAP-ENV:Body>
  <employee>
```

```
<name>Bob</name>
  <phone>827 644 5674</phone>
<name>Susan</name>
  <phone>827 644 5674</phone>
</employee>
</SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```



---

---

## Oracle Application Server Web Services

This chapter describes the Oracle Application Server Web Services features, architecture, and implementation.

This chapter covers the following topics:

- [Oracle Application Server OC4J \(J2EE\) and Oracle SOAP Based Web Services](#)
- [Oracle Application Server Web Services Standards](#)
- [Oracle Application Server Web Services Features](#)
- [Oracle Application Server Web Services Architecture](#)
- [Understanding WSDL and Client Proxy Stubs for Web Services](#)
- [Web Services Home Page](#)
- [About Universal Description, Discovery, and Integration Registry](#)

### Oracle Application Server OC4J (J2EE) and Oracle SOAP Based Web Services

Oracle Application Server supports two different Web Services options, a J2EE based Web Services environment built into Oracle Application Server Containers for J2EE (OC4J), and an Apache SOAP based Web Services environment called Oracle Application Server SOAP.

The chapters in this manual describe the OC4J (J2EE) Web Services environment. This environment makes it easy to develop and deploy services using J2EE artifacts, and is moving the Oracle Application Server Web Services features toward the evolving Web Services standards included in the next release of J2EE (J2EE 1.4). The Oracle Application Server Web Services environment includes many development and deployment features that are integrated with the advanced Oracle Application Server features.

[Appendix A, "Using Oracle Application Server SOAP"](#) describes the Oracle Application Server support for Apache SOAP (Oracle Application Server SOAP). Oracle Application Server includes support for Apache SOAP because this implementation was one of the earliest SOAP implementations and it supports existing Web Services applications.

---

---

**Note:** Oracle recommends using the Oracle Application Server OC4J (J2EE) Web Services environment for developing Web Services. The Apache SOAP (Oracle Application Server SOAP) implementation is currently in maintenance mode.

---

---

## Oracle Application Server Web Services Standards

Oracle Application Server Web Services supports the following Web Services standards:

- SOAP 1.1, including the following:
  - RPC/Encoded
  - Document/Literal
- WSDL 1.1
- UDDI 2.0

**See Also:** ["Overview of Web Services Standards"](#) on page 2-1

## Oracle Application Server Web Services Features

Oracle Application Server provides advanced runtime features and comprehensive support for developing and deploying Web Services. The Oracle Application Server infrastructure includes support for the following:

- [Developing End-to-End Web Services](#)
- [Deploying and Managing Web Services](#)
- [Using Oracle JDeveloper with Web Services](#)
- [Securing Web Services](#)
- [Aggregating Web Services](#)

## Developing End-to-End Web Services

Oracle Application Server Web Services provides comprehensive support for developing Web Services, including:

- **Development Environment** – Oracle Application Server Web Services allows application developers to implement Web Services using J2EE components. In addition, you can use Java Classes or PL/SQL Stored Procedures to implement Web Services. Web Services inherit all the runtime and lifecycle management elements of J2EE Applications.
- **Development Tools and Wizards** – Oracle Application Server Web Services Developers can use the same set of command line utilities to create, package, and deploy Web Services as other Oracle Application Server Containers for J2EE (OC4J) applications.
- **Automatically Generating WSDL** – Oracle Application Server Web Services can generate WSDL and client-side proxy stubs. This generation occurs when the Web Service is assembled using the WebServices Assembly tool or alternatively, for a deployed Web Service, the first time the WSDL or the client-side proxy stubs are requested (after the first request, the previously generated WSDL or client-side proxy stubs are sent when requested).
- **Registration, Publishing, and Discovery** – Oracle Application Server Web Services provides a standards-compliant UDDI registry where Web Services can be published and discovered. The Oracle UDDI registry supports both a private and public UDDI registry and can also synchronize information with other UDDI nodes.

- **Developer Simplicity** – Using Oracle Application Server Web Services, developers do not need to learn a completely new set of concepts – Web Services are developed, deployed and managed using the same programming concepts and tools as with J2EE Applications.
- **Business Logic Reuse** – Application developers can transparently publish their J2EE Applications to new Web Services clients with no change in the application itself. Their existing business logic developed in J2EE can be transparently accessed from existing J2EE/EJB clients or from a new Web Service client.
- **Common Runtime Services** – Oracle Application Server has a common runtime and brokering environment for J2EE Applications and Web Services. As a result, Web Services transparently inherit various services available with the J2EE Container including Transaction Management, Messaging, Naming, Logging, and Security Services.

## Deploying and Managing Web Services

Oracle Enterprise Manager and the Web Services Assembly Tool assist with deploying and managing Oracle Application Server Web Services. These tools provide the following support for Web Services:

- **Packaging and Assembly** - The Web Services Assembly Tool assists with assembling Web Services and producing a J2EE .ear file.
- **Deployment** – Oracle Enterprise Manager provides a comprehensive set of facilities to deploy Web Services to Oracle Application Server. Oracle Enterprise Manager provides a single, consistent *Deploy Applications* wizard for deploying Web Services to Oracle Application Server. It accepts a J2EE .ear file, and walks you through a set of steps to get information about the application to be deployed, and then deploys the application.
- **Register Web Service** - The *Deploy Applications* wizard is only available when deploying Web Services. This step provides access to facilities for registering Web Services in the UDDI Registry.
- **Browse the UDDI Registry** - Oracle's UDDI Registry provides the UDDI standards compliant pre-defined, hierarchical categorization schemes. Oracle Enterprise Manager can drill-down through these categories and look up specific Web Services registered in any category.
- **Monitoring and Administration** – Once deployed, Oracle Enterprise Manager provides facilities to de-install a Web Service and also to monitor Web Service performance, as measured by response-time and throughput, and status, as measured by up-time, CPU, and memory consumption. Oracle Enterprise Manager also provides facilities to identify and list all the Web Services deployed to a specific Oracle Application Server instance.

## Using Oracle JDeveloper with Web Services

The Oracle JDeveloper IDE supports Oracle Application Server Web Services. Oracle JDeveloper is the industry's most advanced Java and XML IDE and provides unparalleled productivity and end-to-end J2EE and integrated Web Services standards compliance.

Oracle JDeveloper supports Oracle Application Server Web Services with the following features:

- Allows developers to create Java stubs from Web Services WSDL descriptions to programmatically use existing Web Services.

- Allows developers to create a new Web Service from Java or EJB classes, automatically producing the required deployment descriptor, web.xml, and WSDL file for you.
- Provides schema-driven WSDL file editing.
- Offers significant J2EE deployment support for Web Services J2EE .ear files, with automatic deployment to OC4J.

## Securing Web Services

Oracle Enterprise Manager secures Oracle Application Server Web Services in the same way that it secures J2EE Servlets running under OC4J. This provides a comprehensive set of security facilities, including:

- Complete, standards-based security architecture for encryption, authentication, and authorization of Web Services.
- Single Sign-on to enable users to access several Web Services with a single password.
- Single Point of administration to enable users to centrally manage the security for Web Services.

## Aggregating Web Services

OracleAS Portal facility provides the ability to aggregate Oracle Application Server Web Services within an organization into a Portal. Additionally, portlets in the OracleAS Portal framework can be published as Web Services.

## Oracle Application Server Web Services Architecture

Oracle Application Server Containers for J2EE (OC4J) provides the foundation for building applications as components and supports Oracle Application Server Web Services. Oracle Application Server Web Services supports both RPC Style and Document Style web services.

Oracle Application Server Web Services supports the following RPC Web Services:

- Java Classes
- Stateless Session Enterprise Java Beans (EJBs)
- Stateless PL/SQL Stored Procedures or Functions

Oracle Application Server Web Services supports the following Document Style web services:

- Java Class Document Style Web Services
- JMS Document Style Web Services

For each implementation type, Oracle Application Server Web Services uses a different Servlet that conforms to J2EE standards to provide an entry point to a Web Service implementation. [Figure 2-1](#) shows the Web Services runtime architecture, including the Servlet entry points.

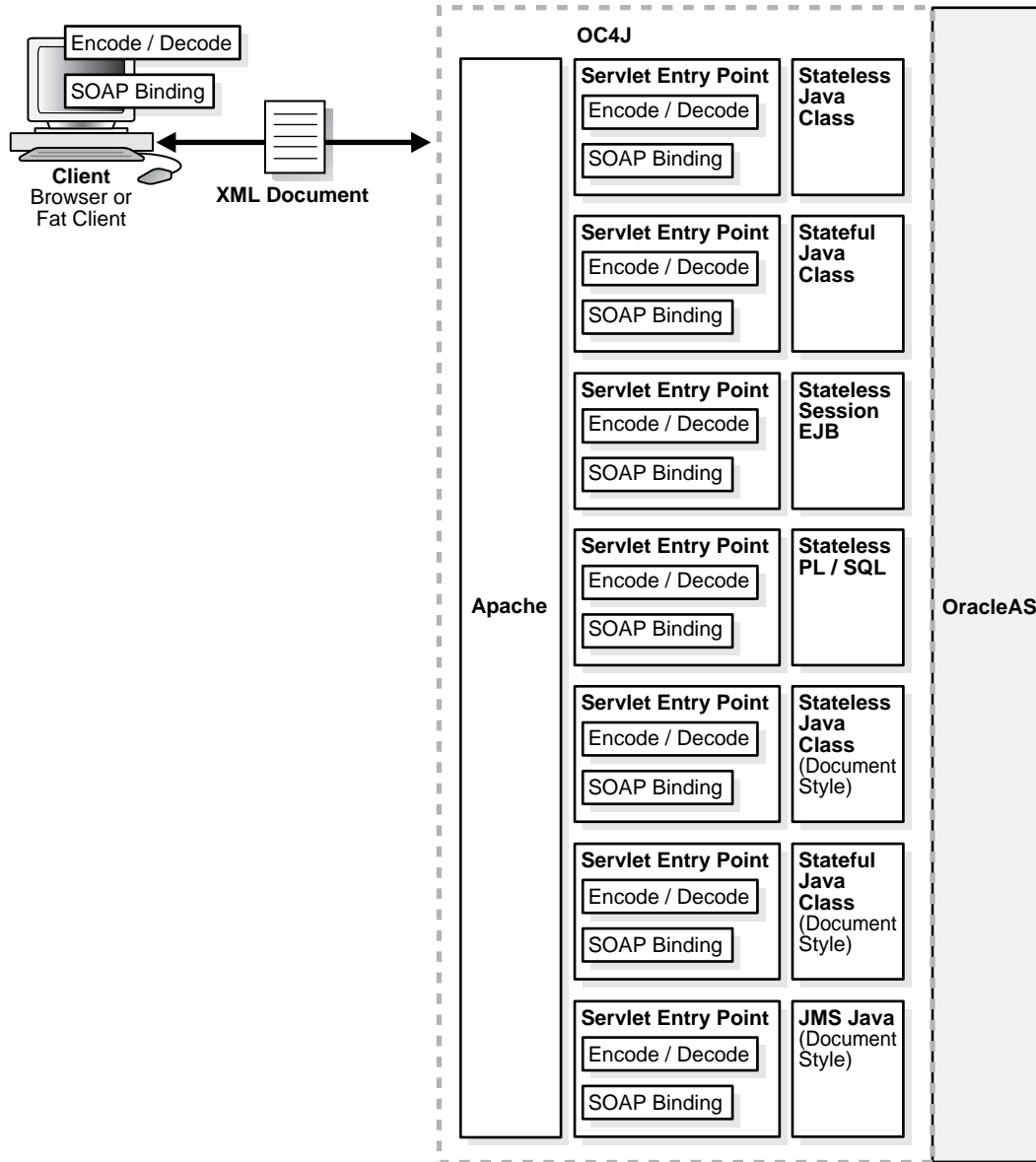
The Oracle Application Server Web Services runtime architecture discussion includes the following:

- [About Servlet Entry Points for Web Services](#)
- [What Are the Packaging and Deployment Options for Web Services](#)

■ [About Server Skeleton Code Generation for Web Services](#)

**See Also:** "SOAP Standard" on page 1-4 for information on RPC Style and Document Style Web Services.

**Figure 2-1 Web Services Runtime Architecture (RPC and Document Style with Servlet Entry Points)**



**About Servlet Entry Points for Web Services**

To use Oracle Application Server Web Services, you need to deploy a J2EE .ear file to Oracle Application Server. The J2EE .ear file contains a Web Services Servlet configuration and includes an implementation of the Web Service. Oracle Application Server Web Services supplies the Servlet classes, one for each supported implementation type. At runtime, Oracle Application Server uses the Servlet classes to access the user supplied Web Service implementation.

The Oracle Application Server Web Services Servlet classes support the following Web Services implementation types:

- Java Class (Stateless) - The object implementing the Web Service is any arbitrary Java class. The Web Service is stateless.
- Java Class (Stateful) -The object implementing the Web Service is any arbitrary Java class. The Web Service is considered stateful. A Servlet `HttpSession` maintains the object state between requests from the same client.
- Stateless Session EJBs - Stateless Session EJBs can be exposed as Web Services. The Web Service is considered to be stateless.
- PL/SQL Stored Procedure or Function - The object implementing the Web Service is a Java class that accesses the PL/SQL stored procedure or function. The Web Service is considered to be stateless. The Oracle JPublisher tool generates the Java access class for the PL/SQL stored procedure or function.
- Java Class Document Style Web Service (Stateless) - The object implementing the Web Service is a Java class using a supported method signature. The Web Service is stateless.
- Java Class Document Style Web Service (Stateful) -The object implementing the Web Service is a Java class using a supported method signature. The Web Service is considered stateful. A Servlet `HttpSession` maintains the object state between requests from the same client.
- Java JMS Web Service - Supports sending and receiving messages to or from JMS destinations. Using the JMS Web Service you can include an MDB to handle or generate messages.

When a Web Service is deployed, a unique instance of the Servlet class manages the Web Service. The Servlet class is implemented as part of Oracle Application Server Web Services runtime support. To make Web Services accessible, you deploy the Web Service implementation with the corresponding Web Services Servlet.

---

---

**Note:** Using Oracle Application Server SOAP, based on Apache SOAP 2.3.1, there is only a single instance of a single Servlet entry point for all the Web Services in the entire system. The Oracle Application Server Web Services architecture differs; under Oracle Application Server Web Services, a unique Servlet instance supports each Web Service.

---

---

RPC Style Web Service implementations under Oracle Application Server Web Services that take values as parameters or that return values to a client need to restrict the types passed. This restriction allows the types passed to be converted between XML and Java objects (and between Java objects and XML). [Table 2-1](#) lists the supported types for passing to or from Oracle Application Server Web Services.

Document Style Web Service implementations under Oracle Application Server Web Services restrict the signature of the Java methods that implement the Web Service. Only `org.w3c.dom.Element` can be passed to or sent from these Web Services.

---

---

**Note:** The preceding restriction means that `org.w3c.dom.Element` types cannot be mixed as a parameter with other types in methods that implement a Web Service.

---

---

**Table 2–1 Web Services Supported Data Types (for RPC Parameters and Return Values)**

Primitive Type	Object Type
Boolean	java.lang.Boolean
byte	java.lang.Byte
double	java.lang.Double
float	java.lang.Float
int	java.lang.Integer
long	java.lang.Long
short	java.lang.Short
string	java.lang.String
	java.util.Date
	java.util.Map
	org.w3c.dom.Element
	org.w3c.dom.Document
	org.w3c.dom.DocumentFragment
	Java Beans (whose property types are listed in this table or are another supported Java Bean)
	Single-dimensional arrays of types listed in this table

## What Are the Packaging and Deployment Options for Web Services

Oracle Application Server Web Services are accessed as Servlets, thus, Web Services need to be assembled. The `WebServicesAssembler` tool prepares J2EE .ear files for Web Services by configuring a `web.xml` file that is a component of a J2EE .war file, and including the required resources and the implementation and support classes.

To build a Web Service with the assembly tool, you can supply a Jar file, .war file, `ejb.jar`, or .ear file that includes your Web Service implementation. The assembly tool then builds the Web Service using configuration information specified in its XML configuration file.

### See Also:

- [Chapter 3, "Developing and Deploying Java Class Web Services"](#)
- [Chapter 4, "Developing and Deploying EJB Web Services"](#)
- [Chapter 5, "Developing and Deploying Stored Procedure Web Services"](#)
- [Chapter 6, "Developing and Deploying Document Style Web Services"](#)

## About Server Skeleton Code Generation for Web Services

The first time Oracle Application Server Web Services receives a request for a service, the Servlet entry point automatically does the following (this discussion does not apply for JMS Web Services, which are handled differently):

- Validates the class loading. All the classes that are required for the Web Service implementation must conform to standard J2EE class loading norms.
- Validates the data types. All the Java classes or EJBs must conform to the restrictions on supported parameter and return types as shown in [Table 2-1](#).
- Generates server skeleton code. The server skeleton code is only generated the first time the Web Service is accessed or when the ear file is redeployed (when an application is redeployed, the server skeleton code and other Web Services support files are regenerated). The generated code is stored in the temporary directory associated with the Servlet context. The server skeleton code controls the lifecycle of the EJB (for Stateless Session EJB implementations), handles the marshaling of the parameters and return types (for SOAP RPC based Web Services), and dispatches to the actual Java class or EJB methods that implement the service.

After the server skeleton class is generated, when subsequent requests for a service are received, the server skeleton directly handles marshalling and then invokes the method that implements the service (for Web Services implemented with PL/SQL stored procedures or functions, the server skeleton invokes the Java class that accesses the Database containing the PL/SQL stored procedure or function).

For document style Web Services, the server skeleton passes the DOM element to the method that implements the service.

## Understanding WSDL and Client Proxy Stubs for Web Services

Oracle Application Server Web Services provides a tool to generate a WSDL file that can be packaged with a Web Service at assembly time, (if you do not package the WSDL file, it can be generated at runtime). This tool also supports generating client-side proxy stubs, given a WSDL file.

There are several elements to Oracle Application Server Web Services WSDL support. First, RPC style Web Services are based on interoperable XML data representations and arbitrary Java objects do not in general map to XML. Oracle Application Server Web Services supports a set of XML types corresponding to a set of Java types (see [Table 2-1](#) for the list of supported Java types).

Second, using Oracle Application Server Web Services, an application developer can either statically generate the WSDL interfaces for a Web Service or the Oracle Application Server Web Services runtime can generate WSDL and client-side proxy stubs if they are not provided when a Web Service is deployed. These files can be generated by the runtime on the server-side and delivered when they are requested by a Web Services client.

Oracle Application Server also provides a client-side tool to statically generate WSDL given a Java class or a J2EE application. Likewise, the Web Services Assembly tool can generate the client-side proxy given a generated WSDL file or a known WSDL endpoint.

### See Also:

- ["Generating Client-Side Proxies With WebServicesAssembler"](#) on page 8-5
- ["Generating WSDL Files and Client Side Proxies"](#) on page 9-3



## Overview of a WSDL Based Web Service Client

Using Web Services, a client application sends a SOAP request that invokes a Web Service and handles the SOAP response from the service. To facilitate client application development, the Oracle Application Server Web Services runtime can generate WSDL to describe a Web Service. Using the WSDL, development tools can assist developers in building applications that invoke Web Services.

**See Also:**

- ["Using Oracle JDeveloper with Web Services"](#) on page 2-3
- [Chapter 8, "Building Clients that Use Web Services"](#)

## Overview of a Client-Side Proxy Stubs Based Web Service Client

Using Web Services, a client application sends a SOAP request that invokes a Web Service and handles the SOAP response from the service. To facilitate client-side application development, Oracle Application Server Web Services can generate client-side proxy stubs. The client-side proxy stubs hide the details of composing a SOAP request and decomposing the SOAP response. The generated client-side proxy stubs support a synchronous invocation model for requests and responses. The generated stubs make it easier to write a Java client application to make a Web Service (SOAP) request and handle the response.

**See Also:** [Chapter 8, "Building Clients that Use Web Services"](#)

## Web Services Home Page

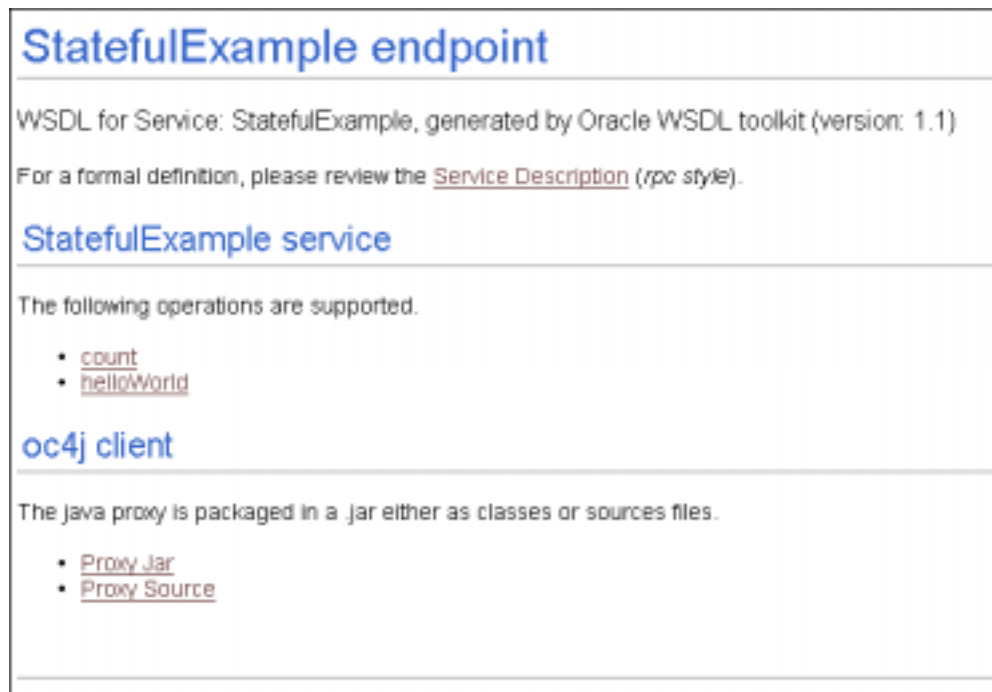
Oracle Application Server Web Services provides a Web Service Home Page for each deployed Web Service.

A Web Service Home Page provides the following:

- A Link to the WSDL file - To obtain the WSDL file for a Web Service, select the Service Description link and save the file.
- Links to Web Service Test Pages for each supported operation-To test the available Web Service operations enter the parameter values for the operation, if any, and select the Invoke button.
- Links to the Web Service client-side proxy Jar and the client-side proxy source - To obtain the client-side proxy Jar or the client-side proxy source, select the appropriate link, Proxy Jar or Proxy Source, and save the file.

[Figure 2-2](#) shows a sample Web Service Home Page.

Figure 2–2 Web Service Home Page



## About Universal Description, Discovery, and Integration Registry

The Universal Description, Discovery, and Integration (UDDI) specification consists of a four-tier hierarchical XML schema that provides the base information model to publish, validate, and invoke information about Web Services. The four types of information that the UDDI XML schema defines are:

- **Business Entity** - The top level XML element in a UDDI entry captures the starting set of information required by partners seeking to locate information about a business' services including its name, its industry or product category, its geographic location, and optional categorization and contact information. This includes support for *Yellow Pages* taxonomies to search for businesses by industry, product, or geography.
- **Business Service** - The businessService structure groups a series of related Web Services together so that they can be related to either a business process or a category of services. An example of a business process could be a logistics/delivery process which could include several Web Services including shipping, routing, warehousing, and last-mile delivery services. By organizing Web Services into groups associated with categories or business processes, UDDI allows more efficient search and discovery of Web Services.
- **Binding Information** - Each businessService has one or more technical Web Service Descriptions captured in an XML element called a binding template. The binding template contains the information that is relevant for application programs that need to invoke or to bind to a specific Web Service. This information includes the Web Service URL address, and other information describing hosted services, routing and load balancing facilities.
- **Compliance Information** - While the bindingTemplate contains the information required to invoke a service, it is not always enough to simply know where to contact a particular Web Service. For instance, to send a business partner's Web

Service a purchase order, the invoking service must not only know the location/URL for the service, but what format the purchase order should be sent in, what protocols are appropriate, what security required, and what form of a response will result after sending the purchase order. Before invoking a Web Service, it is useful to determine whether the specific service being invoked complies with a particular behavior or programming interface. Each bindingTemplate element, therefore, contains an element called a tModel that contains information which enables a client to determine whether a specific Web Service is a compliant implementation.

## **Oracle Enterprise Manager Features to Register Web Services**

When a Web Service is deployed on Oracle Application Server, you can use Oracle Enterprise Manager to register the specific Web Service and publish its WSDL to the UDDI registry and to discover published Web Services.

**See Also:** [Chapter 10, "Discovering and Publishing Web Services"](#)



---

---

# Developing and Deploying Java Class Web Services

This chapter describes the procedures you use to write and deploy Oracle Application Server Web Services that are implemented as Java classes.

This chapter covers the following topics:

- [Using Oracle Application Server Web Services With Java Classes](#)
- [Writing Java Class Based Web Services](#)
- [Preparing and Deploying Java Class Based Web Services](#)
- [Serializing and Encoding Parameters and Results for Web Services](#)

## Using Oracle Application Server Web Services With Java Classes

This chapter shows sample code for writing Web Services implemented with Java classes and describes the difference between writing stateful and stateless Java Web Services.

Oracle Application Server supplies Servlets to access the Java classes which implement a Web Service. The Servlets handle requests generated by a Web Service client, run the Java method that implements the Web Service and returns results back to Web Services clients.

**See Also:**

- [Chapter 2, "Oracle Application Server Web Services"](#)
- [Chapter 4, "Developing and Deploying EJB Web Services"](#)
- [Chapter 5, "Developing and Deploying Stored Procedure Web Services"](#)
- [Chapter 8, "Building Clients that Use Web Services"](#)

## Writing Java Class Based Web Services

Writing Java class based Web Services involves building a Java class that includes one or more methods. When a Web Services client makes a service request, Oracle Application Server Web Services invokes a Web Services Servlet that runs the method that implements the service request. There are very few restrictions on what actions Web Services can perform. At a minimum, Web Services generate some data that is sent to a client or perform an action as specified by a Web Service request.

This section shows how to write a stateful and a stateless Java Web Service that returns a string, "Hello World". The stateful service also returns an integer running count of the number of method calls to the service. This Java Web Service receives a client request and generates a response that is returned to the Web Service client.

The sample code is supplied on the Oracle Technology Network Web site,

<http://otn.oracle.com/tech/java/oc4j/demos/1012/index.html>

After expanding the Web Services demo.zip file, the Java class based Web Service is in the directory under `webservices/demo/basic/java_services` on UNIX or in `\webservices\demo\basic\java_services` on Windows.

## Writing Stateless and Stateful Java Web Services

Oracle Application Server Web Services supports stateful and stateless implementations for Java classes running as Web Services, as follows:

- For a stateful Java implementation, Oracle Application Server Web Services uses a single Java instance to serve the Web Service requests from an individual client.
- For a stateless Java implementation, Oracle Application Server Web Services creates multiple instances of the Java class in a pool, any one of which may be used to service a request. After servicing the request, the object is returned to the pool for use by a subsequent request.

---

---

**Note:** It is the job of the Web Services developer to make the design decision to implement a stateful or stateless Web Service. When packaging Web Services, stateless and stateful Web Services are handled slightly differently. This chapter describes these differences in the section, "[Preparing and Deploying Java Class Based Web Services](#)" on page 3-6.

---

---

## Building a Sample Java Class Implementation

Developing a Java Web Service consists of the following steps:

- [Defining a Java Class Containing Methods for the Web Service](#)
- [Defining an Interface for Explicit Method Exposure](#)
- [Writing a WSDL File \(Optional\)](#)

### Defining a Java Class Containing Methods for the Web Service

Create a Java Web Service by writing or supplying a Java class with methods that are deployed as a Web Service. In the sample supplied in the `java_services` sample directory, the .ear file, `ws_example.ear` contains the Web Service source, class, and configuration files. In the expanded .ear file, the class `StatefulExampleImpl` provides the stateful Java service and `StatelessExampleImpl` provides the stateless Java service.

When writing a Java Web Service, if you want to place the Java service in a package, use the Java package specification to name the package. The first line of `StatefulExampleImpl.java` specifies the package name, as follows:

```
package oracle.j2ee.ws_example;
```

The stateless sample Web Service is implemented with `StatelessExampleImpl`, a public class. The class defines a public method, `helloWorld()`. In general, a Java class for a Web Service defines one or more public methods.

[Example 3-1](#) shows `StatelessExampleImpl`.

The stateful sample Web Service is implemented with `StatefulExampleImpl`, a public class. The class initializes the count and defines two public methods, `count()` and `helloWorld()`.

[Example 3-2](#) shows `StatefulExampleImpl`.

**Example 3-1 Defining A Public Class with Java Methods for a Stateless Web Service**

```
package oracle.j2ee.ws_example;

public class StatelessExampleImpl {
    public StatelessExampleImpl() {
    }
    public String helloWorld(String param) {
        return "Hello World, " + param;
    }
}
```

**Example 3-2 Defining a Public Class with Java Methods for a Stateful Web Service**

```
package oracle.j2ee.ws_example;

public class StatefulExampleImpl {
    int count = 0;
    public StatefulExampleImpl() {
    }
    public int count() {
        return count++;
    }
    public String helloWorld(String param) {
        return "Hello World, " + param;
    }
}
```

A Java class implementation for a Web Service must include a public constructor that takes no arguments. [Example 3-1](#) shows the public constructor `StatelessExampleImpl()` and [Example 3-2](#) shows `StatefulExampleImpl()`.

When an error occurs while running a Web Service implemented as a Java class, the Java class should throw an exception. When an exception is thrown, the Web Services Servlet returns a Web Services (SOAP) fault. Use the standard J2EE and OC4J administration facilities to view the logs of Servlet errors for a Web Service that uses Java classes for its implementation.

When you create a Java class containing methods that implement a Web Service, the method's parameters and return values must use supported types, or you need to use an interface class to limit the methods exposed to those methods using only supported types. [Table 3-1](#) lists the supported types for parameters and return values for Java methods that implement Web Services.

---

**Note:** See [Table 3-1](#) for the list of supported types for parameters and return values.

---

There are several additional steps required to implement a Java Web Service if you need to handle or process SOAP request header entries.

**See Also:** ["SOAP Header Support"](#) on page 12-3

### Defining an Interface for Explicit Method Exposure

Oracle Application Server Web Services allows you to limit the methods you expose as Web Services by supplying a public interface. To limit the methods exposed in a Web Service, include a public interface that lists the method signatures for the methods that you want to expose. [Example 3-3](#) shows an interface to the method in the class `StatelessExampleImpl`. [Example 3-4](#) shows an interface to the methods in the class `StatefulExampleImpl`.

#### **Example 3-3 Using a Public Interface to Expose Stateless Web Services Methods**

```
package oracle.j2ee.ws_example;

public interface StatelessExample {
    String helloWorld(String param);
}
```

#### **Example 3-4 Using a Public Interface to Expose Stateful Web Services Methods**

```
package oracle.j2ee.ws_example;

public interface StatefulExample {
    int count();
    String helloWorld(String param);
}
```

When an interface class is not included with a Web Service, the Web Services deployment exposes all public methods defined in the Java class. Using an interface, for example `StatelessExample` shown in [Example 3-3](#) or `StatefulExample` shown in [Example 3-4](#), exposes only the methods listed in the interface.

---

---

**Note:** Using an interface, only the methods with the specified method signatures are exposed when the Java class is prepared and deployed as a Web Service.

---

---

Use a Web Services interface for the following purposes:

1. To limit the exposure of methods to a subset of the public methods within a class.
2. To expand the set of methods that are exposed as Web Services to include methods within the superclass of a class.
3. To limit the exposure of methods to a subset of the public methods within a class, where the subset contains only the methods that use supported types for parameters or return values. [Table 3-1](#) lists the supported types for parameters and return values for Java methods that implement Web Services.

**See Also:** ["Using Supported Data Types for Java Web Services"](#) on page 3-5

### Writing a WSDL File (Optional)

The `WebServicesAssembler` supports the `<wsdl-gen>` and `<proxy-gen>` tags to allow a Web Service developer to generate WSDL files and client-side proxy files. You



can use these tags to control whether the WSDL file and the client-side proxy are generated. Using these tags you can also specify that the generated WSDL file or a WSDL file that you write is packaged with the Web Service J2EE .ear.

A client-side developer either uses the WSDL file that is obtained from a deployed Web Service, or the client-side proxy that is generated from the WSDL to build an application that uses the Web Service.

**See Also:** ["Generating WSDL Files and Client Side Proxies"](#) on page 9-3

## Using Supported Data Types for Java Web Services

[Table 3-1](#) lists the supported data types for parameters and return values for Oracle Application Server Web Services.

**Table 3-1** *Web Services Supported Data Types*

Primitive Type	Object Type
Boolean	java.lang.Boolean
byte	java.lang.Byte
double	java.lang.Double
float	java.lang.Float
int	java.lang.Integer
long	java.lang.Long
short	java.lang.Short
string	java.lang.String
	java.util.Date
	java.util.Map
	org.w3c.dom.Element
	org.w3c.dom.Document
	org.w3c.dom.DocumentFragment
	Java Beans (whose property types are listed in this table or are another supported Java Bean)
	Single-dimensional arrays of types listed in this table.

Document Style Web Service implementations under Oracle Application Server Web Services restrict the signature of the Java methods that implement the Web Service. Only `org.w3c.dom.Element` can be passed to or sent from these Web Services.

---

**Note:** The preceding restriction means that `org.w3c.dom.Element` types cannot be mixed as a parameter with other types in methods that implement a Web Service.

---



---

**Note:** Oracle Application Server Web Services does not support `Element[]`, (arrays of `org.w3c.dom.Element`).

---

A Bean, for purposes of Web Services, is any Java class which conforms to the following restrictions:

- It must have a constructor taking no arguments.
- It must expose all interesting state through properties.
- It must not matter what order the accessors for the properties, for example, the `setX` or `getX` methods, are in.

Oracle Application Server Web Services allows Beans to be returned or passed in as arguments to J2EE Web Service methods, as long as the Bean only consists of property types that are listed in [Table 3-1](#) or are another supported Java Bean.

When Java Beans are used as parameters to Oracle Application Server Web Services, the client-side code should use the generated Bean included with the downloaded client-side proxy. This is because the generated client-side proxy code translates SOAP structures to and from Java Beans by translating SOAP structure namespaces to and from fully qualified Bean class names. If a Bean with the specified name does not exist in the specified package, the generated client code will fail.

However, there is no special requirement for clients using Web Services Description Language (WSDL) to form calls to Oracle Application Server Web Services, rather than the client-side proxy. The generated WSDL document describes SOAP structures in a standard way. Application development environments, such as Oracle JDeveloper, which work directly from WSDL documents can correctly call Oracle Application Server Web Services with Java Beans as parameters.

---

---

**Note:** When Web Service proxy classes and WSDL are generated, all Java primitive types in the service implementation on the server-side are mapped to Object types in the proxy code or in the WSDL. For example, when the Web Service implementation includes parameters of primitive Java type `int`, the equivalent parameter in the proxy is of type `java.lang.Integer`. This mapping occurs for all primitive types.

---

---

**See Also:** [Chapter 8, "Building Clients that Use Web Services"](#)

## Preparing and Deploying Java Class Based Web Services

To deploy a Java class as a Web Service you need to assemble a J2EE `.ear` file that includes the deployment descriptors for the Oracle Application Server Web Services Servlet and includes the Java class that supplies the Java implementation. This section describes how to use the Oracle Application Server Web Services tool, `WebServicesAssembler`. `WebServicesAssembler` takes an XML configuration file that describes the Java Class Web Service and produces a J2EE `.ear` file that can be deployed under Oracle Application Server Web Services.

This section contains the following topics.

- [Creating a Configuration File to Assemble Java Class Web Services](#)
- [Running WebServicesAssembler To Prepare Java Class Web Services](#)

### Creating a Configuration File to Assemble Java Class Web Services

The Oracle Application Server Web Services assembly tool, `WebServicesAssembler`, assists in assembling Oracle Application Server Web

Services. This section describes how to create a configuration file to use with Java Class Web Services.

Create a `WebServicesAssembler` configuration file by adding the following:

- [Adding Web Service Top Level Tags](#)
- [Adding Java Stateless Service Tags](#)
- [Adding Java Stateful Service Tags](#)
- [Adding WSDL and Client-Side Proxy Generation Tags](#)

### Adding Web Service Top Level Tags

[Table 3–2](#) describes the top level `WebServicesAssembler` configuration file tags. Add these tags to provide top level information describing the Java Stateless Web Service or a Java Stateful Web Service. These tags are included within a `<web-service>` tag in the configuration file.

[Example 3–5](#) shows a complete `config.xml` file, including the top level tags.

**Table 3–2** Top Level `WebServicesAssembler` Configuration Tags

Tag	Description
<code>&lt;context&gt;</code> <i>context</i> <code>&lt;/context&gt;</code>	Specifies the context root of the Web Service. This tag is required.
<code>&lt;datasource-JNDI-name&gt;</code>  <code>&lt;/datasource-JNDI-name&gt;</code>	Specifies the datasource associated with the Web Service.
<code>&lt;description&gt;</code> <i>description</i> <code>&lt;/description&gt;</code>	Provides a simple description of the Web Service. This tag is optional.
<code>&lt;destination-path&gt;</code> <i>dest_path</i> <code>&lt;/destination-path&gt;</code>	Specifies the name of the generated J2EE .ear file output. The <i>dest_path</i> specifies the complete path for the output file. This tag is required.
<code>&lt;display-name&gt;</code> <i>disp_name</i> <code>&lt;/display-name&gt;</code>	Specifies the Web Service display name. This tag is optional.
<code>&lt;option name="source-path"</code> <code>[contextroot="<i>path1</i>" ] &gt;</code> <i>path2</i> <code>&lt;option&gt;</code>	Includes a specified file in the output .ear file. Use this option to specify java resources, or the name of an existing .war, .ear, or ejb-jar file that is used as a source file for the output J2EE .ear file. When a .war file is supplied as input, the optional contextroot specifies the root-context for the .war file. <i>path1</i> specifies the context-root for the .war. <i>path2</i> specifies the path to the file to include. For example: <code>&lt;option name="source-path"</code> <code>contextroot="/test"&gt;/myTestArea/ws/src/statefull.war&lt;/option&gt;</code>

**Table 3–2 (Cont.) Top Level WebServicesAssembler Configuration Tags**

Tag	Description
<code>&lt;stateless-java-service&gt;</code> <i>sub-tags</i> <code>&lt;/stateless-java-service&gt;</code>	Use this tag to add a Java Web Services that defines a stateless service. See <a href="#">Table 3–3</a> for a description of valid <i>sub-tags</i> .
<code>&lt;stateful-java-service&gt;</code> <i>sub-tags</i> <code>&lt;/stateful-java-service&gt;</code>	Use this tag to add a Java Web Services that defines a stateful service. See <a href="#">Table 3–3</a> for a description of valid <i>sub-tags</i> .
<code>&lt;temporary-directory&gt;</code> <i>temp_dir</i> <code>&lt;/temporary-directory&gt;</code>	Specifies a directory where the assembler can store temporary files. This tag is optional.

### Adding Java Stateless Service Tags

Prepare Java Stateless Web Services using the `WebServicesAssembler` `<stateless-java-service>` tag. This tag is included within a `<web-service>` tag in the configuration file. Add this tag to provide information required for generating a Stateless Java Web Service.

[Table 3–3](#) shows the `<stateless-java-service>` *sub-tags* and the `<stateful-java-service>` *sub-tags*. As noted in [Table 3–3](#), some of the *sub-tags* listed only apply when using a `<stateful-java-service>`.

[Example 3–5](#) shows a complete `config.xml` file, including `<stateless-java-service>`.

---

**Note:** It is the job of the Web Services developer to make the design decision to implement a stateful or stateless Web Service. When packaging Web Services, stateless and stateful Web Services are handled slightly differently.

---

### Adding Java Stateful Service Tags

Prepare Java Stateful Web Services using the `WebServicesAssembler` `<stateful-java-service>` tag. This tag is included within a `<web-service>` tag in the configuration file. Add this tag to provide information required for generating a Stateful Java Web Service.

To support a clustered environment, for stateful Java Web Services with serializable java classes, the `WebServicesAssembler` adds a `<distributable>` tag in the `web.xml` of the Web Service's generated J2EE.ear file.

[Table 3–3](#) shows the `<stateful-java-service>` *sub-tags*.

[Example 3–5](#) shows a complete `config.xml` file, including `<stateful-java-service>`.

**Table 3–3 Stateless and Stateful Java Service Sub-Tags**

Tag	Description
<pre>&lt;accept-untyped-request&gt; value &lt;/accept-untyped-request&gt;</pre>	<p>Setting <i>value</i> to true tells <code>WebServicesAssembler</code> to allow the Web Service to accept untyped requests. When the value is false, the Web Service does not accept untyped-request.</p> <p>Valid values: true, false (case is not significant; TRUE and FALSE are also valid)</p> <p>This tag is optional.</p> <p>Default value: false</p>
<pre>&lt;class-name&gt; class &lt;/class-name&gt;</pre>	<p>Specifies the fully qualified class name for the class that supplies the Web Service implementation.</p> <p>This tag is required</p>
<pre>&lt;interface-name&gt; interface &lt;/interface-name&gt;</pre>	<p>Specifies the fully qualified name of the interface that tells the Web Service Servlet generation code which methods should be exposed as Web Services.</p> <p>This tag is optional</p>
<pre>&lt;ejb-resource&gt; ejb-resource &lt;/ejb-resource&gt;</pre>	<p>This is a backward compatibility tag.</p> <p>See Also: the top level <code>&lt;option name="source-path"&gt;</code> tag in <a href="#">Table 3–2</a>.</p> <p>This tag is optional</p>
<pre>&lt;java-resource&gt; resource &lt;/java-resource&gt;</pre>	<p>This is a backward compatibility tag.</p> <p>See Also: the top level <code>&lt;option name="source-path"&gt;</code> tag in <a href="#">Table 3–2</a>.</p> <p>This tag is optional</p>
<pre>&lt;message-style&gt; rpc &lt;/message-style&gt;</pre>	<p>Sets the message style. When defining a Java Web Service, if you include the <code>&lt;message-style&gt;</code> tag you must specify the value <code>rpc</code>.</p> <p>Valid Values: doc, rpc</p> <p>This tag is optional</p> <p>Default value: rpc (when the <code>&lt;message-style&gt;</code> tag is not supplied)</p>
<pre>&lt;scope&gt; scope &lt;/scope&gt;</pre>	<p>Sets the scope of the session for stateful services.</p> <p>The <code>&lt;scope&gt;</code> tag only applies for stateful services. Use this tag only within the <code>&lt;stateful-java-service&gt;</code> tag.</p> <p>This tag is optional</p> <p>Valid Values: application, session</p> <p>Default Value: session</p>
<pre>&lt;session-timeout&gt; value &lt;/session-timeout&gt;</pre>	<p>Sets the session timeout for a stateful session.</p> <p>The <code>&lt;session-timeout&gt;</code> tag only applies for stateful services. Use this tag only within the <code>&lt;stateful-java-service&gt;</code> tag.</p> <p>Specify <i>value</i> with an integer that defines the timeout for the session in seconds. The default value for the session timeout for stateful Java sessions where no session timeout is specified is 60 seconds.</p> <p>This tag is optional</p>
<pre>&lt;uri&gt; URI &lt;/uri&gt;</pre>	<p>Specifies servlet mapping pattern for the Servlet that implements the Web Service. The path specified as the <i>URI</i> is appended to the <code>&lt;context&gt;</code> to specify the Web Service location.</p> <p>This tag is required</p>

**Example 3-5 Sample WebServicesAssembler Configuration File**

```

<web-service>
  <display-name>Web Services Example</display-name>
  <description>Java Web Service Example</description>
  <!-- Specifies the resulting web service archive will be stored in
        ./ws_example.ear -->
  <destination-path>./ws_example.ear</destination-path>
  <!-- Specifies the temporary directory that web service assembly
        tool can create temporary files. -->
  <temporary-directory>./tmp</temporary-directory>
  <!-- Specifies the web service will be accessed in the servlet context
        named "/webservices". -->
  <context>/webservices</context>

  <!-- Specifies the web service will be stateless -->
  <stateless-java-service>
    <interface-name>oracle.j2ee.ws_example.StatelessExample</interface-name>
    <class-name>oracle.j2ee.ws_example.StatelessExampleImpl</class-name>
    <!-- Specifies the web service will be accessed in the uri named
          "statelessTest" within the servlet context. -->
    <uri>/statelessTest</uri>
    <!-- Specifies the location of Java class files are under
          ./src -->
    <java-resource>./src</java-resource>
  </stateless-java-service>

  <stateful-java-service>
    <interface-name>oracle.j2ee.ws_example.StatefulExample</interface-name>
    <class-name>oracle.j2ee.ws_example.StatefulExampleImpl</class-name>
    <!-- Specifies the web service will be accessed in the uri named
          "statefullTest" within the servlet context. -->
    <uri>/statefulTest</uri>
    <!-- Specifies the location of Java class files are under
          ./src -->
    <java-resource>./src</java-resource>
  </stateful-java-service>
</web-service>

```

**Adding WSDL and Client-Side Proxy Generation Tags**

The `WebServicesAssembler` supports the `<wsdl-gen>` and `<proxy-gen>` tags to allow a Web Service developer to generate WSDL files and client-side proxy files. You can use these tags to control whether the WSDL file and the client-side proxy are generated. Using these tags you can also specify that the generated WSDL file or a WSDL file that you supply is packaged with the Web Service J2EE .ear.

A client-side developer can use the WSDL file that is obtained from a deployed Web Service, or the client-side proxy that is generated from the WSDL to build an application that uses the Web Service.

**See Also:** ["Generating WSDL Files and Client Side Proxies"](#) on page 9-3

**Running WebServicesAssembler To Prepare Java Class Web Services**

After you create the `WebServicesAssembler` configuration file, you can generate a J2EE .ear file for the Web Service. The J2EE .ear file includes the Java Web Service servlet configuration information, including the file `web.xml`, and the Java classes and interfaces that you supply.

Run the Oracle Application Server Web Services assembly tool, `WebServicesAssembler` as follows:

```
java -jar WebServicesAssembler.jar -config config_file
```

Where: *config\_file* is the configuration file that contains the `<stateless-java-service>` or the `<stateful-java-service>` tags.

**See Also:**

- ["Creating a Configuration File to Assemble Java Class Web Services"](#) on page 3-6
- ["Running the Web Services Assembly Tool"](#) on page 9-1

## Deploying Java Class Based Web Services

After creating the J2EE .ear file containing the Java classes and the Web Services Servlet deployment descriptors you can deploy the Web Service as you would any standard J2EE application stored in an .ear file (to run under OC4J).

**See Also:** *Oracle Application Server Containers for J2EE User's Guide* in the Oracle Application Server 10g Documentation Library

## Serializing and Encoding Parameters and Results for Web Services

Parameters and results sent between Web Service clients and a Web Service implementation go through the following steps:

1. Parameters are serialized and encoded in XML when sent from the Web Service client.
2. Parameters are deserialized and decoded from XML when the Web Service receives a request on the server side.
3. Parameters or results are serialized and encoded in XML when a request is returned from a Web Service to a Web Service client.
4. Parameters or results must be deserialized and decoded from XML when the Web Service client receives a reply.

Oracle Application Server Web Services supports a prepackaged implementation for handling these four steps for serialization and encoding, and deserialization and decoding. The prepackaged mechanism makes the four serialization and encoding steps transparent both for the Web Services client-side application, and for the Java service writer that is implementing a Web Service. Using the prepackaged mechanism, Oracle Application Server Web Services supports the following encoding mechanisms:

- **Standard SOAP v.1.1 encoding:** Using standard SOAP v.1.1 encoding, the server side Web Services Servlet that calls the Java class implementation handles serialization and encoding internally for the types supported by Oracle Application Server Web Services. [Table 3-1](#) lists the supported Web Services parameter and return value types when using standard SOAP v.1.1 encoding.
- **Literal XML encoding.** Using Literal XML encoding, a Web Service client can pass as a parameter, or a Java service can return as a result, a value that is encoded as a conforming W3C Document Object Model (DOM) `org.w3c.dom.Element`. When an `Element` passes as a parameter to a Web Service, the server side Java implementation processes the `org.w3c.dom.Element`. For return values sent

from a Web Service, the Web Services client parses or processes the `org.w3c.dom.Element`.

---

---

**Note:** For parameters to a Web Service or results that the Web Service generates and returns to Web Services clients, the Oracle Application Server Web Services implementation supports either the Standard SOAP encoding or Literal XML encoding but not both, for any given Web Service (Java method).

---

---

**See Also:** [Chapter 8, "Building Clients that Use Web Services"](#)



---

---

## Developing and Deploying EJB Web Services

This chapter describes the procedures you use to write and deploy Oracle Application Server Web Services that are implemented as stateless session Enterprise Java Beans (EJBs).

This chapter covers the following topics:

- [Using Oracle Application Server Web Services With Stateless Session EJBs](#)
- [Writing Stateless Session EJB Web Services](#)
- [Preparing and Deploying Stateless Session EJB Based Web Services](#)

### Using Oracle Application Server Web Services With Stateless Session EJBs

This chapter shows sample code for writing Web Services implemented with stateless session EJBs.

Oracle Application Server supplies Servlets to access the EJBs which implement a Web Service. A Servlets handle requests generated by a Web Service client, locates the EJB home and remote interfaces, runs the EJB that implements the Web Service, and returns results back to the Web Service client.

**See Also:**

- [Chapter 2, "Oracle Application Server Web Services"](#)
- [Chapter 3, "Developing and Deploying Java Class Web Services"](#)
- [Chapter 5, "Developing and Deploying Stored Procedure Web Services"](#)
- [Chapter 8, "Building Clients that Use Web Services"](#)

### Writing Stateless Session EJB Web Services

Writing EJB based Web Services involves obtaining or building an EJB that implements a service. The EJB should contain one or more methods that a Web Services Servlet running under Oracle Application Server invokes when a client makes a Web Services request. There are very few restrictions on what actions Web Services can perform. At a minimum, Web Services usually generate data that is sent to a Web Services client or perform an action as specified by a Web Services method request.

This section shows how to write a simple stateless session EJB Web Service, `HelloService` that returns a string, "Hello World", to a client. This EJB Web Service

receives a client request with a single `String` parameter and generates a response that it returns to the Web Service client.

The sample code is supplied on the Oracle Technology Network Web site,

<http://otn.oracle.com/tech/java/oc4j/demos/1012/index.html>

After expanding the Web Services `demo.zip` file, the EJB based Web Service is in the directory under `/webservices/demo/basic/stateless_ejb` on UNIX or in `\webservices\demo\basic\stateless_ejb` on Windows.

Create a stateless session EJB Web Service by writing a standard J2EE stateless session EJB containing a remote interface, a home interface, and an enterprise bean class. Oracle Application Server Web Services runs EJBs that are deployed as Oracle Application Server Web Services in response to a request issued by a Web Service client.

Developing a stateless session EJB consists of the following steps:

- [Defining a Stateless Session Remote Interface](#)
- [Defining a Stateless Session Home Interface](#)
- [Defining a Stateless Session EJB Bean](#)
- [Returning Results From EJB Web Services](#)
- [Error Handling for EJB Web Services](#)
- [Serializing and Encoding Parameters and Results for EJB Web Services](#)
- [Using Supported Data Types for Stateless Session EJB Web Services](#)
- [Writing a WSDL File for EJB Web Services \(Optional\)](#)

**See Also:** ["Preparing and Deploying Stateless Session EJB Based Web Services"](#) on page 4-6

## Defining a Stateless Session Remote Interface

When looking at the `HelloService` EJB Web Service, note that the `.ear` file, `HelloService.ear` defines the Web Service and its configuration files. In the sample directory, the file `HelloService.java` provides the remote interface for the `HelloService` EJB.

[Example 4-1](#) shows the `Remote` interface for the sample stateless session EJB.

### **Example 4-1** Stateless Session EJB Remote Interface for Web Service

```
package demo;

public interface HelloService extends javax.ejb.EJBObject {
    java.lang.String hello(java.lang.String phrase) throws java.rmi.RemoteException;
}
```

## Defining a Stateless Session Home Interface

The sample file `HelloServiceHome.java` provides the home interface for the `HelloService` EJB.

[Example 4-2](#) shows the `EJBHome` interface for the sample stateless session EJB.

**Example 4–2 Stateless Session EJB Home Interface for Web Service**

```

package demo;
/**
 * This is a Home interface for the Session Bean
 */
public interface HelloServiceHome extends javax.ejb.EJBHome {

    HelloService create() throws javax.ejb.CreateException, java.rmi.RemoteException
    ;
}

```

**Defining a Stateless Session EJB Bean**

The sample file `HelloServiceBean.java` provides the Bean logic for the `HelloService` EJB. When you create a Bean to implement a Web Service, the parameters and return values must be of supported types. [Table 4–1](#) lists the supported types for parameters and return values for stateless session EJBs that implement Web Services.

[Example 4–3](#) shows the source code for the `HelloService` Bean.

**Example 4–3 Stateless Session EJB Bean Class for Web Services**

```

package demo;

import java.rmi.RemoteException;
import java.util.Properties;
import javax.ejb.*;

/**
 * This is a Session Bean Class.
 */
public class HelloServiceBean implements SessionBean {
    private javax.ejb.SessionContext mySessionCtx = null;

    public void ejbActivate() throws java.rmi.RemoteException {}
    public void ejbCreate() throws javax.ejb.CreateException, java.rmi.RemoteException
    {}

    public void ejbPassivate() throws java.rmi.RemoteException {}
    public void ejbRemove() throws java.rmi.RemoteException {}
    public javax.ejb.SessionContext getSessionContext() {
        return mySessionCtx;
    }
    public String hello(String phrase)
    {
        return "HELLO!! You just said :" + phrase;
    }
    public void setSessionContext(javax.ejb.SessionContext ctx) throws
    java.rmi.RemoteException {
        mySessionCtx = ctx;
    }
}

```

## Returning Results From EJB Web Services

The `hello()` method shown in [Example 4-3](#) returns a `String`. An Oracle Application Server Web Services server-side Servlet runs the Bean that calls the `hello()` method when the Servlet receives a Web Services request from a client. After executing the `hello()` method, the Servlet returns a result to the Web Service client.

[Example 4-3](#) shows that the EJB Bean writer only needs to return values of supported types to create Web Services implemented as stateless session EJBs.

**See Also:** ["Using Supported Data Types for Stateless Session EJB Web Services"](#) on page 4-4

## Error Handling for EJB Web Services

When an error occurs while running a Web Service implemented as an EJB, the EJB should throw an exception. When an exception is thrown, the Web Services Servlet returns a Web Services (SOAP) fault. Use the standard J2EE and OC4J administration facilities for logging Servlet errors for a Web Service that uses stateless session EJBs for its implementation.

## Serializing and Encoding Parameters and Results for EJB Web Services

Parameters and results sent between Web Service clients and a Web Service implementation need to be encoded and serialized. This allows the call and return values to be passed as XML documents using SOAP.

**See Also:** ["Serializing and Encoding Parameters and Results for Web Services"](#) on page 3-11

## Using Supported Data Types for Stateless Session EJB Web Services

[Table 4-1](#) lists the supported data types for parameters and return values for Oracle Application Server Web Services.

**Table 4-1** *Web Services Supported Data Types*

Primitive Type	Object Type
Boolean	<code>java.lang.Boolean</code>
byte	<code>java.lang.Byte</code>
double	<code>java.lang.Double</code>
float	<code>java.lang.Float</code>
int	<code>java.lang.Integer</code>
long	<code>java.lang.Long</code>
short	<code>java.lang.Short</code>
string	<code>java.lang.String</code>
	<code>java.util.Date</code>
	<code>java.util.Map</code>
	<code>org.w3c.dom.Element</code>
	<code>org.w3c.dom.Document</code>
	<code>org.w3c.dom.DocumentFragment</code>

**Table 4–1 (Cont.) Web Services Supported Data Types**

Primitive Type	Object Type
	Java Beans (whose property types are listed in this table or are another supported Java Bean)
	Single-dimensional arrays of types listed in this table.

---

**Note:** Oracle Application Server Web Services does not support `Element[]`, (arrays of `org.w3c.dom.Element`).

---

Document Style Web Service implementations under Oracle Application Server Web Services restrict the signature of the Java methods that implement the Web Service. Only `org.w3c.dom.Element` can be passed to or sent from these Web Services.

---

**Note:** The preceding restriction means that `org.w3c.dom.Element` types cannot be mixed as a parameter with other types in methods that implement a Web Service.

---

A Bean, for purposes of Web Services, is any Java class which conforms to the following restrictions:

- It must have a constructor taking no arguments.
- It must expose all interesting state through properties.
- It must not matter what order the accessors for the properties, for example, the `setX` or `getX` methods, are in.

Oracle Application Server Web Services allows Beans to be returned or passed in as arguments to J2EE Web Service methods, as long as the Bean only consists of property types that are listed in [Table 4–1](#) or are another supported Java Bean.

When Java Beans are used as parameters to Oracle Application Server Web Services, the client-side code should use the generated Bean included with the downloaded client-side proxy. This is because the generated client-side proxy code translates SOAP structures to and from Java Beans by translating SOAP structure namespaces to and from fully qualified Bean class names. If a Bean with the specified name does not exist in the specified package, the generated client code will fail.

However, there is no special requirement for clients using Web Services Description Language (WSDL) to form calls to Oracle Application Server Web Services, rather than the client-side proxy. The generated WSDL document describes SOAP structures in a standard way. Application development environments, such as Oracle JDeveloper, which work directly from WSDL documents can correctly call Oracle Application Server Web Services with Java Beans as parameters.

---

**Note:** When Web Service proxy classes and WSDL are generated, all Java primitive types in the service implementation on the server-side are mapped to Object types in the proxy code or in the WSDL. For example, when the Web Service implementation includes parameters of primitive Java type `int`, the equivalent parameter in the proxy is of type `java.lang.Integer`. This mapping occurs for all primitive types.

---

**See Also:** [Chapter 8, "Building Clients that Use Web Services"](#)

## Writing a WSDL File for EJB Web Services (Optional)

The `WebServicesAssembler` supports the `<wsdl-gen>` and `<proxy-gen>` tags to allow a Web Service developer to generate WSDL files and client-side proxy files. You can use these tags to control whether the WSDL file and the client-side proxy are generated. Using these tags you can also specify that the generated WSDL file or a WSDL file that you write is packaged with the Web Service J2EE .ear.

A client-side developer either uses the WSDL file that is obtained from a deployed Web Service, or the client-side proxy that is generated from the WSDL to build an application that uses the Web Service.

**See Also:** ["Generating WSDL Files and Client Side Proxies"](#) on page 9-3

## Preparing and Deploying Stateless Session EJB Based Web Services

To deploy a stateless session EJB as a Web Service you need to assemble a J2EE .ear file that includes the deployment descriptors for the Oracle Application Server Web Services Servlet and includes the `ejb.jar` that supplies the Java implementation. This section describes how to use the Oracle Application Server Web Services tool, `WebServicesAssembler`. `WebServicesAssembler` takes an XML configuration file that describes the stateless session EJB Web Service and produces a J2EE .ear file that can be deployed under Oracle Application Server Web Services.

This section contains the following topics.

- [Creating a Configuration File to Assemble Stateless Session EJB Web Services](#)
- [Running WebServicesAssembler To Prepare Stateless Session EJB Web Services](#)
- [Deploying Web Services Implemented as EJBs](#)

### Creating a Configuration File to Assemble Stateless Session EJB Web Services

The Oracle Application Server Web Services assembly tool, `WebServicesAssembler`, assists in assembling Oracle Application Server Web Services. This section describes how to create a configuration file to use with stateless session EJB Web Services.

Create `WebServicesAssembler` configuration file by adding the following:

- [Adding Web Service Top Level Tags](#)
- [Adding Stateless Session EJB Service Tags](#)
- [Adding WSDL and Client-Side Proxy Generation Tags](#)

#### Adding Web Service Top Level Tags

[Table 4-2](#) describes the top level `WebServicesAssembler` configuration file tags. Add these tags to provide top level information describing the Java Stateless Web Service or a Java Stateful Web Service. These tags are included within a `<web-service>` tag in the configuration file.

[Example 4-4](#) shows a complete `config.xml` file, including the top level tags.

**Table 4–2** Top Level `WebServicesAssembler` Configuration Tags

Tag	Description
<code>&lt;context&gt;</code> <i>context</i> <code>&lt;/context&gt;</code>	Specifies the context root of the Web Service. This tag is required.
<code>&lt;datasource-JNDI-name&gt;</code> <i>datasource</i> <code>&lt;/datasource-JNDI-name&gt;</code>	Specifies the datasource associated with the Web Service.
<code>&lt;description&gt;</code> <i>description</i> <code>&lt;/description&gt;</code>	Provides a simple description of the Web Service. This tag is optional.
<code>&lt;destination-path&gt;</code> <i>dest_path</i> <code>&lt;/destination-path&gt;</code>	Specifies the name of the generated J2EE .ear file output. The <i>dest_path</i> specifies the complete path for the output file. This tag is required.
<code>&lt;display-name&gt;</code> <i>disp_name</i> <code>&lt;/display-name&gt;</code>	Specifies the Web Service display name. This tag is optional.
<code>&lt;option name="source-path"&gt;</code> <i>path</i> <code>&lt;/option&gt;</code>	Includes a specified file in the output .ear file. Use this option to specify java resources, or the name of an existing .war, .ear, or ejb-jar file that is used as a source file for the output J2EE .ear file.  When a .war file is supplied as input, the optional <i>contextroot</i> specifies the root-context for the .war file.  <i>path1</i> specifies the context-root for the .war. <i>path2</i> specifies the path to the file to include.  For example: <pre>&lt;option name="source-path" contextroot="/test"&gt;/myTestArea/ws/src/statefull.war &lt;/option&gt;</pre> This tag is optional.
<code>&lt;stateless-session-ejb-service&gt;</code> <i>sub-tags</i> <code>&lt;/stateless-session-ejb-service&gt;</code>	Use this tag to add a stateless session EJB Web Service. See <a href="#">Table 4–3</a> for a description of the valid <i>sub-tags</i> .
<code>&lt;temporary-directory&gt;</code> <i>temp_dir</i> <code>&lt;/temporary-directory&gt;</code>	Specifies a directory where the assembler can store temporary files. This tag is optional.

### Adding Stateless Session EJB Service Tags

Prepare Stateless Session EJB Web Services using the `WebServicesAssembler` `<stateless-session-ejb-service>` tag. This tag is included within a `<web-service>` tag in the configuration file. Add this tag to provide information required for generating a stateless session EJB Web Service.

[Table 4–3](#) shows the `<stateless-session-ejb-service>` sub-tags.

[Example 4–4](#) shows a complete `config.xml` file, including `<stateless-session-ejb-service>`.

**Table 4–3 Stateless Session EJB Web Service Sub-Tags**

Tag	Description
<code>&lt;accept-untyped-request&gt;</code> <i>value</i> <code>&lt;/accept-untyped-request&gt;</code>	Setting <i>value</i> to true tells <code>WebServicesAssembler</code> to allow the Web Service to accept untyped requests. When the value is false, the Web Service does not accept untyped-request.  Valid values: true, false  (case is not significant; TRUE and FALSE are also valid)  This tag is optional.  Default value: false
<code>&lt;ejb-name&gt;</code> <i>name</i> <code>&lt;/ejb-name&gt;</code>	Specifies the <i>name</i> of the stateless session EJB.  This tag is required
<code>&lt;ejb-resource&gt;</code> <i>resource</i> <code>&lt;/ejb-resource&gt;</code>	This is a backward compatibility tag.  See Also: the top level <code>&lt;option name="source-path"&gt;</code> tag in <a href="#">Table 4–2</a> .  This tag is optional
<code>&lt;path&gt;</code> <i>path</i> <code>&lt;/path&gt;</code>	This is a backward compatibility tag.  See Also: the top level <code>&lt;option name="source-path"&gt;</code> tag in <a href="#">Table 4–2</a> .  This tag is optional
<code>&lt;uri&gt;</code> <i>URI</i> <code>&lt;/uri&gt;</code>	Specifies servlet mapping pattern for the Servlet that implements the Web Service. The path specified as the <i>URI</i> is appended to the <code>&lt;context&gt;</code> to specify the Web Service location.  This tag is required.

**Example 4–4 Sample Stateless Session EJB WebServicesAssembler Configuration File**

```

<web-service>
  <display-name>EJB Web Services Demo</display-name>
  <destination-path>tmp/HelloService.ear</destination-path>
  <temporary-directory>tmp</temporary-directory>
  <context>/sejb_webservices</context>

  <stateless-session-ejb-service>
    <path>tmp/Hello.jar</path>
    <uri>/HelloService</uri>
    <ejb-name>HelloService</ejb-name>
  </stateless-session-ejb-service>
</web-service>

```

**Adding WSDL and Client-Side Proxy Generation Tags**

The `WebServicesAssembler` supports the `<wsdl-gen>` and `<proxy-gen>` tags to allow a Web Service developer to generate WSDL files and client-side proxy files. You can use these tags to control whether the WSDL file and the client-side proxy are generated. Using these tags you can also specify that the generated WSDL file or a WSDL file that you write is packaged with the Web Service J2EE .ear.

A client-side developer either uses the WSDL file that is obtained from a deployed Web Service, or the client-side proxy that is generated from the WSDL to build an application that uses the Web Service.



**See Also:** ["Generating WSDL Files and Client Side Proxies"](#) on page 9-3

## Running WebServicesAssembler To Prepare Stateless Session EJB Web Services

After you create the `WebServicesAssembler` configuration file, you can generate a J2EE `.ear` file for the Web Service. The J2EE `.ear` file includes the stateless session EJB Web Service servlet configuration information.

Run the Oracle Application Server Web Services assembly tool, `WebServicesAssembler` as follows:

```
java -jar WebServicesAssembler.jar -config config_file
```

Where: *config\_file* is the configuration file that contains the `<stateless-session-ejb-service>` tag.

**See Also:**

- ["Creating a Configuration File to Assemble Stateless Session EJB Web Services"](#) on page 4-6
- ["Running the Web Services Assembly Tool"](#) on page 9-1

## Deploying Web Services Implemented as EJBs

After creating the `.ear` file containing a stateless session EJB, you can deploy the Web Service as you would any standard J2EE application stored in an `.ear` file (to run under OC4J).

**See Also:** *Oracle Application Server Containers for J2EE User's Guide* in the Oracle Application Server Documentation Library



---

---

# Developing and Deploying Stored Procedure Web Services

This chapter describes how to write and deploy Oracle Application Server Web Services implemented as stateless PL/SQL Stored Procedures or Functions (**Stored Procedure Web Services**). Stored Procedure Web Services enable you to export, as services running under Oracle Application Server Web Services, PL/SQL procedures and functions that run on an Oracle database server.

This chapter covers the following topics:

- [Using Oracle Application Server Web Services with Stored Procedures](#)
- [Writing Stored Procedure Web Services](#)
- [Preparing Stored Procedure Web Services](#)
- [Deploying Stored Procedure Web Services](#)
- [Limitations for Stored Procedures Running as Web Services](#)

## Using Oracle Application Server Web Services with Stored Procedures

This chapter shows sample code for writing Web Services implemented with stateless PL/SQL stored procedures or functions. The sample is based on a PL/SQL package representing a company that manages employees.

Oracle Application Server Web Services supplies a Servlet to access Java classes that support PL/SQL Stored Procedure Web Services. The Servlet handles requests generated by a Web Service client, runs the Java method that accesses the stored procedure that implements the Web Service, and returns results back to the Web Service client.

The Oracle database server supports procedures implemented in languages other than PL/SQL, including Java and C/C++. These stored procedures can be exposed as Web Services using PL/SQL interfaces.

### See Also:

- [Chapter 2, "Oracle Application Server Web Services"](#)
- [Chapter 3, "Developing and Deploying Java Class Web Services"](#)
- [Chapter 6, "Developing and Deploying Document Style Web Services"](#)

## Writing Stored Procedure Web Services

Writing Stored Procedure Web Services involves creating and installing a PL/SQL package on an Oracle database server that is available as a datasource to Oracle Application Server and generating a Java class that includes one or more methods to access the Stored Procedure.

The sample code is supplied on the Oracle Technology Network Web site,

<http://otn.oracle.com/tech/java/oc4j/demos/1012/index.html>

After expanding the Web Services `demo.zip` file, the sample Stored Procedure Web Service is supplied in the directory under `webservices/demo/basic/stored_procedure` on UNIX or in `webservices\demo\basic\stored_procedure` on Windows.

Create a Stored Procedure Web Service by writing and installing a PL/SQL Stored Procedure. To write and install a PL/SQL Stored Procedure, you need to use facilities independent of Oracle Application Server Web Services.

For example, to use the sample `COMPANY` package, first create and load the supplied package on the database server using the `create.sql` script. This script, along with several other required `.sql` scripts are in the `stored_procedure` directory. These scripts create several database tables and the sample `COMPANY` package.

When the Oracle database server is running on the local system, use the following command to create the sample PL/SQL package:

```
sqlplus scott/tiger @create
```

When the Oracle database server is not the local system, use the following command and include a connect identifier to create the sample PL/SQL package:

```
sqlplus scott/tiger@db_service_name @create
```

where `db_service_name` is the net service name for the Oracle database server.

### See Also:

- ["Limitations for Stored Procedures Running as Web Services"](#) on page 5-9
- *PL/SQL User's Guide and Reference* in the Oracle Database Documentation Library
- *Oracle Net Services Administrator's Guide* in the Oracle Database Documentation Library

## Preparing Stored Procedure Web Services

This section describes how to use the Oracle Application Server Web Services tool `WebServicesAssembler` to prepare a J2EE `.ear` file that supports using a PL/SQL procedure or function as a Stored Procedure Web Service.

This section contains the following topics:

- [Creating a Configuration File to Assemble Stored Procedure Web Services](#)
- [Running WebServicesAssembler With Stored Procedure Web Services](#)
- [Setting Up Datasources in Oracle Application Server Web Services \(OC4J\)](#)

## Creating a Configuration File to Assemble Stored Procedure Web Services

The Oracle Application Server Web Services assembly tool, `WebServicesAssembler`, assists in assembling Oracle Application Server Web Services. This section describes how to create a configuration file to use to assemble a Stored Procedure Web Service. The Web Services assembly tool uses an XML configuration file that describes the Stored Procedure Web Service and produces a J2EE `.ear` file that can be deployed under Oracle Application Server Web Services.

Create `WebServicesAssembler` configuration file by adding the following:

- [Adding Web Service Top Level Tags](#)
- [Adding Stateless Stored Procedure Java Service Tags](#)
- [Adding WSDL and Client-Side Proxy Generation Tags](#)

### Adding Web Service Top Level Tags

[Table 5-1](#) describes the top level `WebServicesAssembler` configuration file tags. Add these tags to provide top level information describing the PL/SQL Stored Procedure Web Service.

[Example 5-1](#) shows a complete `config.xml` file, including the top level tags.

**Table 5-1** Top Level `WebServicesAssembler` Configuration Tags

Tag	Description
<code>&lt;context&gt;</code> <i>context</i> <code>&lt;/context&gt;</code>	Specifies the context root of the Web Service. This tag is required.
<code>&lt;datasource-JNDI-name&gt;</code> <i>datasource</i> <code>&lt;/datasource-JNDI-name&gt;</code>	Specifies the datasource associated with the Web Service.
<code>&lt;description&gt;</code> <i>description</i> <code>&lt;/description&gt;</code>	Provides a simple description of the Web Service. This tag is optional.
<code>&lt;destination-path&gt;</code> <i>dest_path</i> <code>&lt;/destination-path&gt;</code>	Specifies the name of the generated J2EE <code>.ear</code> file output. The <i>dest_path</i> specifies the complete path for the output file. This tag is required.
<code>&lt;display-name&gt;</code> <i>disp_name</i> <code>&lt;/display-name&gt;</code>	Specifies the Web Service display name. This tag is optional.
<code>&lt;option name="source-path"&gt;</code> <i>path</i> <code>&lt;/option&gt;</code>	Includes a specified file in the output <code>.ear</code> file. Use this option to include Java resources. The <i>path</i> specifies the path to the file to include.
<code>&lt;stateless-stored-procedure-java-service&gt;</code> <i>sub-tags</i> <code>&lt;/stateless-stored-procedure-java-service&gt;</code>	Use this tag to add stateless stored procedure Web Services. See <a href="#">Table 5-2</a> and <a href="#">Table 5-4</a> for a description of valid <i>sub-tags</i> .
<code>&lt;temporary-directory&gt;</code> <i>temp_dir</i> <code>&lt;/temporary-directory&gt;</code>	Specifies a directory where the assembler can store temporary files. This tag is optional.

## Adding Stateless Stored Procedure Java Service Tags

There are two ways to develop Stored Procedure Web Services using the `WebServicesAssembler`:

- [Adding Stateless Stored Procedure Java Service Using Jar Generation](#)
- [Adding Stateless Stored Procedure Java Services Using a Pre-generated Jar](#)

---

---

**Note:** Most Stored Procedure Web Service developers use the Jar generation technique for assembling the Web Service J2EE .ear file. Only use the pre-generated Jar technique for creating a J2EE .ear when you have a pre-generated Jar file containing Oracle JPublisher generated classes.

---

---

### Adding Stateless Stored Procedure Java Service Using Jar Generation

Using a configuration file that includes the `<jar-generation>` tag specifies Oracle Database Server connection information that allows the `WebServicesAssembler` to run Oracle JPublisher to generate the classes to support the Stored Procedure Web Service. The Oracle JPublisher generated classes support accessing the PL/SQL procedure or function and also includes classes for mapping Java types to PL/SQL types. The `WebServicesAssembler` packages the generated classes into a Jar file that is assembled with the Stored Procedure Web Service.

[Table 5-2](#) describes the `<stateless-stored-procedure-java-service>` `WebServicesAssembler` configuration file tags used when creating a configuration file that uses Jar generation to create a Stored Procedure Web Service. The `<stateless-stored-procedure-java-service>` tag is included within a `<web-service>` tag in the configuration file. Add this tag to provide information required for generating the Stored Procedure Web Service J2EE .ear file.

[Table 5-3](#) describes the sub-tags for `<jar-generation>` within the `<stateless-stored-procedure-java-service>` tag. The `<jar-generation>` tags provide information to the `WebServicesAssembler` so that it can run Oracle JPublisher to generate the Java classes for the Stored Procedure Web Service. The `WebServicesAssembler` then uses these classes to generate the Jar file that provides Java mappings for the stored procedure or function.

[Example 5-1](#) shows a complete `config.xml` file, including the Stored Procedure Web Service tags shown in [Table 5-2](#) and [Table 5-3](#).

**Table 5–2 Stateless Stored Procedure Sub-Tags (Using Jar Generation)**

Tag	Description
<code>&lt;database-JNDI-name&gt;</code> <i>source_JNDI_name</i> <code>&lt;/database-JNDI-name&gt;</code>	This tag specifies the JNDI name of the backend database. The data-sources.xml OC4J configuration file describes the database server source associated with the specified <i>source_JNDI_name</i> .
<code>&lt;jar-generation&gt;</code> <i>sub-tags</i> <code>&lt;/jar-generation&gt;</code>	<a href="#">Table 5-3</a> describes the supported <i>sub-tags</i> for <code>&lt;jar-generation&gt;</code> . Example: <pre>&lt;jar-generation&gt;   &lt;schema&gt;scott/tiger&lt;/schema&gt;   &lt;db-url&gt;jdbc:oracle:thin:@system1:1521:orcl&lt;/db-url&gt;   &lt;prefix&gt;sp.company&lt;/prefix&gt;   &lt;db-pkg-name&gt;Company&lt;/db-pkg-name&gt; &lt;/jar-generation&gt;</pre>
<code>&lt;uri&gt;</code> <i>URI</i> <code>&lt;/uri&gt;</code>	This tag specifies servlet mapping pattern for the Servlet that implements the Web Service. The path specified as the <i>URI</i> is appended to the <code>&lt;context&gt;</code> to specify the Web Service location.

**Table 5–3 Stateless Stored Procedure <jar-generation> Sub-Tags**

Tag	Description
<code>&lt;db-pkg-name&gt;</code> <i>pkg_name</i> <code>&lt;/db-pkg-name&gt;</code>	Where <i>pkg_name</i> is the name of the PL/SQL package to export. This is required when <code>&lt;jar-generation&gt;</code> is included.
<code>&lt;db-url&gt;</code> <i>url_path</i> <code>&lt;/db-url&gt;</code>	Where <i>url_path</i> is the database connect string for the Oracle database server with the specified package to export. The <code>&lt;schema&gt;</code> and <code>&lt;db-url&gt;</code> are combined to connect to the database which contains the stored procedures to be exported. This is required when <code>&lt;jar-generation&gt;</code> is included. Example: <pre>&lt;db-url&gt;jdbc:oracle:thin:@system1.us.oracle.com:1521:tv1&lt;/db-url&gt;</pre>
<code>&lt;method-name&gt;</code> <i>method</i> <code>&lt;/method-name&gt;</code>	Where <i>method</i> is the name of the PL/SQL method to export. This tag is optional. Including multiple <code>&lt;method&gt;</code> tags is valid. In this case the specified methods are exported. Without this tag, all methods within the package are exported. If the specified method is overloaded, then all variations of the method are exported.
<code>&lt;prefix&gt;</code> <i>prefix</i> <code>&lt;/prefix&gt;</code>	Where <i>prefix</i> is the Java package prefix for generated classes. By default, the PL/SQL package is generated into a Java class in the default Java package. This tag is optional. Example: <pre>&lt;prefix&gt;sp.company&lt;/prefix&gt;</pre>
<code>&lt;schema&gt;</code> <i>user_name/password</i> <code>&lt;/schema&gt;</code>	This tag includes the Database Server <i>user_name/password</i> : where: <i>user_name</i> is the database user name. <i>password</i> is the database password for the specified user name. This tag is required when <code>&lt;jar-generation&gt;</code> is included. Example: <pre>&lt;schema&gt;scott/tiger&lt;/schema&gt;</pre>

**Example 5–1 Sample WebServicesAssembler Configuration File For Stored Procedure Using <jar-generation> Tag**

```

<web-service>
  <display-name>Web Services Example</display-name>
  <description>Java Web Service Example</description>
  <!-- Specifies the resulting web service archive will be stored in ./spexample.ear -->
  <destination-path>./spexample.ear</destination-path>
  <!-- Specifies the temporary directory that web service assembly tool can create temporary files. -->
  <temporary-directory>/tmp</temporary-directory>
  <!-- Specifies the web service will be accessed in the servlet context named "/webservices". -->
  <context>/webservices</context>
  <!-- Specifies the web service will be stateless -->

  <stateless-stored-procedure-java-service>
    <jar-generation>
      <schema>scott/tiger</schema>
      <db-url>jdbc:oracle:thin:@system1:1521:orcl</db-url>
      <prefix>sp.company</prefix>
      <db-pkg-name>Company</db-pkg-name>
    </jar-generation>
    <!-- Specifies the web service will be accessed in the uri named
         "statelessSP" within the servlet context. -->
    <uri>/statelessSP</uri>
    <database-JNDI-name>/jdbc/OracleDataSource</database-JNDI-name>
  </stateless-stored-procedure-java-service>
</web-service>

<wsdl-gen>
  <wsdl-dir>wsdl</wsdl-dir>
  <!--force 'true' will write over existing wsdl -->
  <option name="force">true</option>
  <!-- change this to point to your soap servers http listener -->
  <option name="httpServerURL">http://localhost:8888</option>
</wsdl-gen>

<proxy-gen>
  <proxy-dir>proxy</proxy-dir>
  <!-- include-source 'true' will create an additional jar with only the proxy source-->
  <option name="include-source">true</option>
</proxy-gen>
</web-service>

```

**Adding Stateless Stored Procedure Java Services Using a Pre-generated Jar**

Using a configuration file that specifies the stored procedure <class-name> and <interface-name> assembly options when a pre-generated Jar file that includes the required classes to support the Web Service is available. The <class-name> and <interface-name> tags specified in a configuration file support using a previously generated Jar file that contains the Java classes that provide a mapping between the PL/SQL procedure or function and the Web Service.

Table 5–4 describes the <stateless-stored-procedure-java-service> WebServicesAssembler configuration file tags used when creating a configuration file that uses a pre-generated Jar file to create a Stored Procedure Web Service. The <stateless-stored-procedure-java-service> tag is included within a <web-service> tag in the configuration file. Add this tag to provide information required for generating the Stored Procedure Web Service J2EE .ear file.

The <class> and <interface> tags that are added to the <stateless-stored-procedure-java-service> only when using a pre-generated Jar file.



**Table 5–4 Stateless Stored Procedure Sub-Tags (Using Pre-generated Jar File)**

Tag	Description
<pre>&lt;class-name&gt; class &lt;/class-name&gt;</pre>	<p>The Stored Procedure Web Services Servlet definition requires a <code>&lt;param-name&gt;</code> with the value <code>class-name</code> and a corresponding <code>&lt;param-value&gt;</code> set to the fully qualified name of the Java class that accesses the PL/SQL Web Service implementation.</p> <p>You need to use the configuration file <code>&lt;class-name&gt;</code> tag to supply the class name for this parameter; you can find the class name in the Jar file you provide that is specified in the top level <code>&lt;option name="source-path"&gt;</code> tag.</p>
<pre>&lt;database-JNDI-name&gt; source_JNDI_name &lt;/database-JNDI-name&gt;</pre>	<p>This tag specifies the JNDI name of the backend database.</p> <p>The <code>data-sources.xml</code> OC4J configuration file describes the database server source associated with the specified <code>source_JNDI_name</code>.</p>
<pre>&lt;interface-name&gt; interface &lt;/interface-name&gt;</pre>	<p>A Stored Procedure Web Services Servlet definition requires a <code>&lt;param-name&gt;</code> with the value <code>interface-name</code>, and a corresponding <code>&lt;param-value&gt;</code> set to the fully qualified name of the Java interface that specifies the methods to include in the stored procedure Web Service.</p> <p>The <code>&lt;interface-name&gt;</code> tag provides the name of the interface that tells the Web Service Servlet generation code which methods should be exposed as Web Services. You can find the interface name in the Jar file you provide that is specified in the top level <code>&lt;option name="source-path"&gt;</code> tag.</p>
<pre>&lt;java-resource&gt; resource &lt;/java-resource&gt;</pre>	<p>This is a backward compatibility tag.</p> <p>See Also: the top level <code>&lt;option name="source-path"&gt;</code> tag in <a href="#">Table 5–1</a>.</p> <p>This tag is optional.</p> <p>The Stored Procedure pre-generated Jar file should be specified using the <code>&lt;java-resource&gt;</code> tag. The class specified with the <code>&lt;class-name&gt;</code> tag and the interface specified with the <code>&lt;interface-name&gt;</code> tag must exist in the resource specified in the <code>&lt;java-resource&gt;</code> tag(s).</p>
<pre>&lt;uri&gt; URI &lt;/uri&gt;</pre>	<p>This tag specifies servlet mapping pattern for the Servlet that implements the Web Service. The path specified as the <code>URI</code> is appended to the <code>&lt;context&gt;</code> to specify the Web Service location.</p>

**See Also:**

- ["Adding Stateless Stored Procedure Java Service Using Jar Generation"](#) on page 5-4
- *Oracle9i JPublisher User's Guide* in the Oracle Database Documentation Library

**Adding WSDL and Client-Side Proxy Generation Tags**

The `WebServicesAssembler` configuration file supports the `<wsdl-gen>` and `<proxy-gen>` tags to allow a Web Service developer to generate Web Service description WSDL files and client-side proxy files. You can add these tags to control whether the WSDL file and the client-side proxy are generated. You can also specify that the WSDL file be assembled with the Stored Procedure Style Web Service J2EE .ear. A client-side developer can then use the WSDL file that is obtained from the deployed Web Service to build an application that uses the Web Service.

**See Also:** ["Generating WSDL Files and Client Side Proxies"](#) on page 9-3

## Running WebServicesAssembler With Stored Procedure Web Services

After you create the `WebServicesAssembler` configuration file, you can generate a J2EE .ear file for the Stored Procedure Web Service. The J2EE .ear file includes Stored Procedure Web Service servlet configuration information, including the file `web.xml`, and Oracle JPublisher generated classes (the `WebServicesAssembler` collects the Oracle JPublisher generated classes into a single Jar file that it includes in the generated J2EE .ear).

Run the Oracle Application Server Web Services assembly tool, `WebServicesAssembler` as follows:

```
java -jar WebServicesAssembler.jar -config my_pl_service_config
```

Where: `my_pl_service_config` is the configuration file that contains the `<stateless-stored-procedure-java-service>` tag.

### See Also:

- ["Creating a Configuration File to Assemble Stored Procedure Web Services"](#) on page 5-3
- ["Running the Web Services Assembly Tool"](#) on page 9-1

## Setting Up Datasources in Oracle Application Server Web Services (OC4J)

To add Web Services based on PL/SQL Stored Procedures you need to set up data sources in OC4J by configuring `data-sources.xml`. Configuring the `data-sources.xml` file points OC4J to a database. The database should contain PL/SQL Stored Procedure packages that implement a Stored Procedure Web Service.

A single database connection is created when OC4J initializes a Web Services Servlet instance. The resulting database connection is destroyed when OC4J removes the Web Services Servlet instance. Each Stored Procedure Web Services Servlet implements a single threaded model. As a result, any Web Services Servlet instance can only service a single client's database connection requests at any given time. OC4J pools the Web Services Servlet instances and assigns instances to Oracle Application Server Web Services clients.

Every invocation of a PL/SQL Web Service is implicitly a separate database transaction. It is not possible to have multiple service method invocations run within a single database transaction. When such semantics are required, the user must write a PL/SQL procedure that internally invokes other procedures and functions, and then expose the new procedure as another method in a Stored Procedure Web Service (but Oracle Application Server Web Services does not provide explicit support or tools to do this).

When using an emulated data source with CLOB or BLOB types in the stored procedure, the emulated data source must use the `location` attribute to specify the JNDI name. The name cannot be specified using the `ejb-location`.

**See Also:** *Oracle Application Server Containers for J2EE User's Guide* in the Oracle Application Server 10g Documentation Library

## Deploying Stored Procedure Web Services

After creating the J2EE .ear file containing the Stored Procedure Web Service configuration, class, Jar, and support files you can deploy the Web Service as you would any standard J2EE application stored in a J2EE .ear file (to run under OC4J).

**See Also:** *Oracle Application Server Containers for J2EE User's Guide* in the Oracle Application Server 10g Documentation Library

## Limitations for Stored Procedures Running as Web Services

This section covers the following topics:

- [Supported Stored Procedure Features for Web Services](#)
- [Unsupported Stored Procedure Features for Web Services](#)
- [Database Server Release Limitation for Boolean Use in Oracle PL/SQL Web Services](#)
- [TIMESTAMP and DATE Granularity Limitation](#)
- [LOB \(CLOB/BLOB\) Emulated Data Source Limitation](#)

## Supported Stored Procedure Features for Web Services

Stored Procedure Web Services support the following PL/SQL features:

1. PL/SQL stored procedures, including both procedures and functions.
2. IN, OUT, IN, INOUT parameter modes. When a stored procedure contains OUT or INOUT parameters, the INOUT and OUT data are passed back to the client as attributes of the returned objects. The declared stored procedure return value, if the stored procedure is a function, will also be included as an attribute of the returned objects INOUT parameter modes.
3. Packaged procedures only (top-level procedures must be wrapped in a package before they can be exported as a Web Service).
4. Overloaded procedures. Oracle JPublisher may map multiple PL/SQL types into the same Java type. For example, different PL/SQL number types may all map to Java `int`. This means that methods that were considered overloaded in PL/SQL are no longer overloaded in Java. In this case the Java method names will be renamed to avoid compilation errors for the generated code. However, at runtime, the PL/SQL engine may report `PLS-00307` error (too many declarations of `<method name>` match this call). The error is due to PL/SQL limitation on overloading resolution.
5. Simple PL/SQL types

The following simple types are supported. NULL values are supported for all of the simple types listed, except NATURALN and POSITIVEN.

The Oracle JPublisher documentation provides full details on the mappings for these simple types.

VARCHAR2 (STRING, VARCHAR), LONG, CHAR (CHARACTER), NUMBER (DEC, DECIMAL, DOUBLE PRECISION, FLOAT, INTEGER, INT, NUMERIC, REAL, SMALLINT), PLS\_INTEGER, BINARY\_INTEGER (NATURAL, NATURALN, POSITIVE, POSITIVEN), BOOLEAN

6. **TIMESTAMP** is supported, along with variations **TIMESTAMP WITH LOCAL TIME ZONE** and **TIMESTAMP WITH TIME ZONE**.
7. **DATE** is supported.
8. User-defined Object Types.
9. Oracle JPublisher and Oracle Application Server Web Services provide support for the following LOB types: **BLOB**, **CLOB**, and **BFILE**.

If your PL/SQL procedures use LOB types as input/output types, then the WebServices Assembler will not publish those stored procedures that will cause runtime errors. For instance, the WebServices Assembler will not publish a method containing **BFILE** as an **IN** parameter.

10. **SYS.XMLTYPE** is supported. **SYS.XMLTYPE** is mapped into the type, `org.w3c.dom.DocumentFragment` in Web Services.

**See Also:** *Oracle9i JPublisher User's Guide* in the Oracle Database Documentation Library

## Unsupported Stored Procedure Features for Web Services

Stored Procedure Web Services impose the following limitations on PL/SQL functions and procedures:

1. Only procedures and functions within a PL/SQL package are exported as Web Services. Top-level stored procedures must be wrapped inside a package. Methods must be wrapped into package-level methods with a default "this" reference.
2. **NCHAR** and related types are not supported.
3. Oracle JPublisher translates almost all PL/SQL types to Java types. The deployment tools for Stored Procedure Web Services generate "jdbc" style for builtin and number types and "oracle" style for user types and lob types. The lob types are converted to java types that can be serialized/deserialized by Web Services. The user types that conform to java beans are also serialized/deserialized by Web Services. Check the Oracle JPublisher documentation for full details of these styles, and for the caveats associated with them.
4. Fractional seconds in a **TIMESTAMP** value are not preserved when using Stored Procedure Web Services.
5. **TIMESTAMP** as a field in a user defined ADT is not supported. However, **DATE** as a field in a user defined ADT is supported.

**See Also:** *Oracle9i JPublisher User's Guide* in the Oracle Database Documentation Library

## Database Server Release Limitation for Boolean Use in Oracle PL/SQL Web Services

Using a Oracle Database Server of Release 9.2.0.1 or earlier, or with a Database Server that is not Java-enabled, then you must install the **SYS.SQLJUTIL** package into the **SYS** schema to support PL/SQL **BOOLEAN** arguments.

The PL/SQL script that defines this package is located at the following location on UNIX:

```
${ORACLE_HOME}/sqlj/lib/sqljutil.sql
```

On Windows systems, this script is located at the following location:

`%ORACLE_HOME%\sqlj\lib\sqljutil.sql`

### **TIMESTAMP and DATE Granularity Limitation**

Fractional seconds in a `TIMESTAMP` value are not preserved when using Stored Procedure Web Services.

### **LOB (CLOB/BLOB) Emulated Data Source Limitation**

When using an emulated data source with `CLOB` or `BLOB` types, the emulated data source must use the `location` attribute to specify the JNDI name. The name cannot be specified using the `ejb-location`.



---

## Developing and Deploying Document Style Web Services

This chapter describes the procedures you use to write and deploy Oracle Application Server Web Services that handle document style messages and are implemented as Java classes.

This chapter covers the following topics:

- [Using Document Style Web Services](#)
- [Writing Document Style Web Services](#)
- [Preparing Document Style Web Services](#)
- [Deploying Document Style Web Services](#)

### Using Document Style Web Services

This chapter describes Document Style Web Services that are implemented with Java classes and describes the difference between writing stateful and stateless Document Style Java Web Services.

The sample code is supplied on the Oracle Technology Network Web site,

<http://otn.oracle.com/tech/java/oc4j/demos/1012/index.html>

After expanding the Web Services `demo.zip` file, the Document Style Web Services samples are in the `stateless` and `stateful` directories under `webservices/demo/basic/java_doc__services` on UNIX or in `webservices\demo\basic\java_doc_services` on Windows.

Oracle Application Server supplies Servlets to access the Java classes which you write to implement a Web Service. The Servlets handle messages generated by Web Services clients and dispatch them to run the Java methods that implement Document Style Web Services. After a Web Service is deployed, when a client makes a service request (uses a service) the Oracle Application Server Web Services runtime, using an automatically generated Web Services Servlet invokes the methods that you implement to support the Document Style Web Service.

**See Also:**

- [Chapter 3, "Developing and Deploying Java Class Web Services"](#)
- [Chapter 4, "Developing and Deploying EJB Web Services"](#)
- [Chapter 7, "Developing and Deploying JMS Web Services"](#)
- [Chapter 8, "Building Clients that Use Web Services"](#)

## Writing Document Style Web Services

Writing Document Style Java Web Services involves building a Java class that includes one or more methods using supported method signatures; the java class includes methods that either handle an incoming message or return an outgoing message.

This section covers the following topics:

- [Supported Method Signatures for Document Style Web Services](#)
- [Writing Stateless and Stateful Document Style Web Services](#)
- [Writing Classes and Interfaces for Document Style Web Services](#)

### Supported Method Signatures for Document Style Web Services

[Table 6–1](#) shows the supported method signatures for Document Style Web Services. The Oracle Application Server Web Services runtime verifier rejects Document Style Web Services that do not conform to the method signatures listed in [Table 6–1](#).

The `Element` input parameter and `Element` return value shown in the method signatures in [Table 6–1](#) must conform to the Document Object Model (DOM) as specified by the W3C (`org.w3c.dom.Element`).

**Table 6–1 Supported Method Signatures for Document Style Java Web Services**

Method Signature	Description
<code>public Element op_Name(Element e_name)</code>	The method <code>op_Name</code> is a Document Style Web Service operation implemented as a Java method that takes an <code>Element e_name</code> as an input parameter and returns an <code>Element</code> .
<code>public Element get_Name()</code>	The method <code>get_Name</code> is a Document Style Web Service operation implemented as a Java method that takes no input parameters and returns an <code>Element</code> .
<code>public void set_Name(Element e_name)</code>	The method <code>set_Name</code> is a Document Style Web Service operation implemented as a Java method that takes an <code>Element e_name</code> as an input parameter and returns nothing.

### Passing Null Values for Document Style Web Services

A `null` could be passed as an input `Element` or as the `Element` that the Document Style Web Service returns.

### Arrays of Elements

Oracle Application Server Web Services does not support `Element []` (arrays of `org.w3c.dom.Element`).



**See Also:**

- ["Handling Messages for Document Style Web Services"](#) on page 6-6
- <http://www.w3.org/DOM/> for information on the W3C Document Object Model (DOM)

## Writing Stateless and Stateful Document Style Web Services

Oracle Application Server Web Services supports stateful and stateless implementations for Document Style Java classes running as Web Services. For a stateful Java implementation, Oracle Application Server Web Services allows a single Java instance to serve the Web Service requests from an individual client.

For a stateless Java implementation, Oracle Application Server Web Services creates multiple instances of the Java class in a pool, any one of which may be used to service a request. After servicing the request, the object is returned to the pool for use by a subsequent request.

---

---

**Note:** It is the job of the Web Services developer to make the design decision to implement a stateful or stateless Web Service. When packaging Web Services, stateless and stateful Web Services are handled slightly differently. This chapter describes these differences in the section, ["Preparing Document Style Web Services"](#) on page 6-6.

---

---

---

---

**Note:** Deploying a stateful Java implementation class as a stateless Document Style Web Service could yield unpredictable results.

---

---

## Writing Classes and Interfaces for Document Style Web Services

Developing a Document Style Java Web Service consists of the following steps:

- [Defining Methods in a Document Style Web Service](#)
- [Defining an Interface for Explicit Method Exposure](#)
- [Handling Messages for Document Style Web Services](#)

### Defining Methods in a Document Style Web Service

Create a Document Style Web Service by writing or supplying a Java class with methods that are deployed as a Document Style Web Service. The `stateful` and `stateless` sample directories contain sample stateless and stateful Document Style Web Services. In the `src` directories, the file `StatefulDocImpl.java` provides the implementation of the sample stateful Java service and `StatelessDocImpl.java` provides the implementation of the stateless Document Style Web Service. These examples use interface classes; the use of interface classes is optional when implementing Document Style Web Services.

A Java class that implements a Document Style Web Service has the following limitations:

- The Java class should define public methods that conform to the method signatures shown in [Table 6-1](#). If you use an interface, then only the public

methods specified in the interface need to conform to the method signature restrictions. If you do not include an interface, then all the public methods in the class must conform to the method signature restrictions shown in [Table 6-1](#).

- The Java class implementation must include a public constructor that takes no arguments.

There are very few restrictions on what actions a Document Style Java class based web service can perform. At a minimum, the service performs some action to handle an incoming message (`Element`) or to generate an outgoing message (`Element`).

The `StatelessDoc` Web Service sample is implemented with `StatelessDocImpl`, a public class and the interface `StatelessDoc`. The `StatelessDocImpl` class defines two public methods: `displayElement()`, that displays the incoming message on the server where the web service runs, and `processElement()`, that takes an incoming message and returns a transformed message to the client. The private method `applyXSLtoXML()` is a helper method that transforms the incoming message, as specified in the `converter.xml` file.

[Example 6-1](#) shows the method signatures for the `StatelessDocImpl` class (see the `src` directory to view the complete source code for `StatelessDocImpl`).

#### **Example 6-1 Defining Java Methods for a Stateless Document Style Web Service**

```
import org.w3c.dom.*;
import oracle.xml.parser.v2.*;
import java.io.*;

public class StatelessDocImpl implements StatelessDoc

{
    public StatelessDocImpl()
    { }

    // Display the Element that was sent
    public void displayElement(Element e)
    { }

    //method to process the input xml doc
    public Element processElement(Element e)
    { }

    /**
     * This Method Transforms an XML Document into another using the provided
     * Style Sheet: converter.xml. Note : This Method makes use of XSL
     * Transformation capabilities of Oracle XML Parser Version 2.0
     */
    private Element applyXSLtoXML(Element e)
        throws Exception
    {}
}
```

The `StatefulDoc` Web Service sample is implemented with `StatefulDocImpl`, a public class and the interface `StatefulDoc`. The `StatefulDocImpl` class defines two public methods: `startShopping()` that initializes the state of the customer information and `makePurchase()`, that modifies the state of the customer information and returns the updated information to the client. The private method `processElement()` is a helper method that processes the customer's XML element representing a purchase and returns the updated XML element.

[Example 6-2](#) shows the method signatures for the `StatefulDoc` class (see the `src` directory to view the complete source code for `StatefulDocImpl`).

**Example 6-2 Defining Java Methods for a Stateful Document Style Web Service**

```
import org.w3c.dom.*;
import oracle.xml.parser.v2.*;

public class StatefulDocImpl implements StatefulDoc
{
    private Element e ;
    public void startShopping(Element e)
    {
    }
    public Element makePurchase()
    {
    }
    private void processElement(Element e) {
    }
}
```

### Defining an Interface for Explicit Method Exposure

Oracle Application Server Web Services allows you to limit the methods you expose as Document Style Web Services by supplying a public interface. To limit the methods exposed in a Web Service, include a public interface that lists the method signatures for the methods that you want to expose. [Example 6-3](#) shows an interface for the methods in the class `StatelessDocImpl`. [Example 6-4](#) shows an interface for the methods in the class `StatefulDocImpl`.

When an interface is included with a Document Style Web Service, then only the public methods specified in the interface need to conform to the method signature restrictions shown in [Table 6-1](#). If you do not include an interface, then all the public methods in the class must conform to the method signature restrictions. Using an interface, for example `StatelessDoc` shown in [Example 6-3](#), only the methods with the specified method signatures are exposed when the Java class is prepared and deployed as a Document Style Web Service.

Use a Document Style Web Service interface for the following purposes:

1. To limit the exposure of methods to a subset of the public methods within a class.
2. To expand the set of methods that are exposed to include methods within the superclass of a class.
3. To limit the exposure of methods to a subset of the public methods within a class, where the subset contains only the methods that use supported method signatures. [Table 6-1](#) lists the supported signatures for Java methods that implement Document Style Web Services.

**Example 6-3 Using a Public Interface to Expose Stateless Java Services**

```
import org.w3c.dom.*;

public interface StatelessDoc
{
    //method to display the element
    public void displayElement(Element e) ;

    //method to process the input xml doc
    public Element processElement(Element e) ;
}
```

**Example 6–4 Using a Public Interface to Expose Stateful Java Services**

```
import org.w3c.dom.Element;

// Interface that implements getElement and setElement
public interface StatefulDoc {

    // Set the Element
    public void startShopping(Element e);

    // Retrieve the element that was set
    public Element makePurchase();
}
```

**Handling Messages for Document Style Web Services**

It is entirely up to the Web Service developer to determine the processing that occurs for messages associated with a Document Style Web Service.

The message associated with a Document Style Web Service is specified in the `Element` parameter or the `Element` return value associated with the Document Style Web Service. It is the Document Style Web Service developer's job to process or generate messages. The only limitation on Document Style Web Service messages is that the `Element` must conform to must conform to the Document Object Model (DOM) as specified by the W3C (`org.w3c.dom.Element`).

A Document Style Web Service implementation or the client that uses a service may need to supports `null` values, since a `null` could be passed as an input `Element` or as the `Element` that is returned.

For example, the following is valid for a Document Style Web Service implementation:

```
Element get_op () {
    return null;
}
```

## Preparing Document Style Web Services

This section describes how to use the Oracle Application Server Web Services tool `WebServicesAssembler` to prepare a J2EE `.ear` file for a stateless and stateful Document Style Web Service implemented as Java classes.

To deploy a Java class that implements a Document Style Web Service, you need to assemble a J2EE `.ear` file that includes the deployment descriptors for the Oracle Application Server Web Services Servlet and the Java classes that supply the Java implementation. A Web Service implemented with Java classes includes a `.war` file that provides configuration information for the Web Services Servlet running under Oracle Application Server Containers for J2EE (OC4J). This section describes the procedures you use to create a configuration file to use with the `WebServicesAssembler`.

This section contains the following topics:

- [Creating a Configuration File to Assemble Document Style Web Services](#)
- [Running WebServicesAssembler With Document Style Web Services](#)

## Creating a Configuration File to Assemble Document Style Web Services

The Oracle Application Server Web Services assembly tool, `WebServicesAssembler`, assists in assembling Oracle Application Server Web Services. This section describes how to create a configuration file to use to assemble a Document Style Web Service. The Web Services assembly tool uses an XML configuration file that describes the Document Style Web Service. The `WebServicesAssembler` uses the configuration file to produce a J2EE .ear file that can be deployed under Oracle Application Server Web Services.

Create `WebServicesAssembler` configuration file by adding the following:

- [Adding Web Service Top Level Tags](#)
- [Adding Java Service Tags with Document Message Style Specified](#)
- [Adding WSDL and Client-Side Proxy Generation Tags](#)

### Adding Web Service Top Level Tags

[Table 6–2](#) describes the top level `WebServicesAssembler` configuration file tags. Add these tags to provide top level information describing the Document Style Web Service.

[Example 6–5](#) shows a complete stateless sample configuration file. [Example 6–6](#) shows a complete stateful sample configuration file. The `stateless` and `stateful` directories in the `java_doc_services` demo directory contain the sample `config.xml` files.

**Table 6–2** Top Level `WebServicesAssembler` Configuration Tags

Tag	Description
<code>&lt;context&gt;</code> <i>context</i> <code>&lt;/context&gt;</code>	Specifies the context root of the Web Service. This tag is required.
<code>&lt;datasource-JNDI-name&gt;</code> <i>name</i> <code>&lt;/datasource-JNDI-name&gt;</code>	Specifies the datasource associated with the Web Service.
<code>&lt;description&gt;</code> <i>description</i> <code>&lt;/description&gt;</code>	Provides a simple description of the Web Service. This tag is optional.
<code>&lt;destination-path&gt;</code> <i>dest_path</i> <code>&lt;/destination-path&gt;</code>	Specifies the name of the generated J2EE .ear file output. The <i>dest_path</i> specifies the complete path for the output file. This tag is required.
<code>&lt;display-name&gt;</code> <i>disp_name</i> <code>&lt;/display-name&gt;</code>	Specifies the Web Service display name. This tag is optional.

**Table 6–2 (Cont.) Top Level WebServicesAssembler Configuration Tags**

Tag	Description
<code>&lt;option name=source-path"&gt; path &lt;/option&gt;</code>	<p>Includes a specified file in the output .ear file. Use this option to specify java resources, or the name of an existing .war, .ear, or ejb-jar file that is used as a source file for the output J2EE .ear file.</p> <p>When a .war file is supplied as input, the optional contextroot specifies the root-context for the .war file.</p> <p><i>path1</i> specifies the context-root for the .war.</p> <p><i>path2</i> specifies the path to the file to include.</p> <p>For example:</p> <pre>&lt;option name="source-path" contextroot="/test"&gt;/myTestArea/ws/src/statefull.war&lt;/option&gt;</pre> <p>This tag is optional.</p>
<code>&lt;stateless-java-service&gt; sub-tags &lt;/stateless-java-service&gt;</code>	<p>Use this tag to add a Document Style Web Services that defines a stateless service. See <a href="#">Table 6–3</a> for a description of valid <i>sub-tags</i>.</p>
<code>&lt;stateful-java-service&gt; sub-tags &lt;/stateful-java-service&gt;</code>	<p>Use this tag to add a Document Style Web Services that defines a stateful service. See <a href="#">Table 6–3</a> for a description of valid <i>sub-tags</i>.</p>
<code>&lt;temporary-directory&gt; temp_dir &lt;/temporary-directory&gt;</code>	<p>Specifies a directory where the assembler can store temporary files.</p> <p>This tag is optional.</p>

### Adding Java Service Tags with Document Message Style Specified

The Document Style Web Service developer determines if the service is stateful or stateless. The configuration file includes different tags depending on the type of the service. This section covers the tags for both cases, including:

- [Adding Stateful Document Style Java Service Tags](#)
- [Adding Stateless Document Style Java Service Tags](#)

**Table 6–3 Java Service WebServicesAssembler Configuration Tags - Document Style**

Tag	Description
<code>&lt;class-name&gt; value &lt;/class-name&gt;</code>	<p>The Document Style Web Service definition requires at least one <code>&lt;class-name&gt;</code> tag. The <i>value</i> specifies the name of the Java class that provides the Document Style Web Service implementation.</p> <p>This tag is required.</p>
<code>&lt;interface-name&gt; interface &lt;/interface-name&gt;</code>	<p>A Document Style Web Service configuration file supports the optional <code>&lt;interface-name&gt;</code> tag. The corresponding <i>interface</i> value supplied specifies the name of the Java interface that lists the methods to include in the Document Style Web Service.</p> <p>This tag is optional.</p>
<code>&lt;java-resource&gt; resource &lt;/java-resource&gt;</code>	<p>This tag supports adding a Java <i>resource</i>. This specifies the location of the java resources to include in the Document Style Web Service.</p> <p>Include multiple <code>&lt;java-resource&gt;</code> tags to include multiple Java resources.</p> <p>This tag is optional</p>

**Table 6–3 (Cont.) Java Service WebServicesAssembler Configuration Tags - Document Style**

Tag	Description
<pre>&lt;message-style&gt; doc &lt;/message-style&gt;</pre>	<p>When defining a Document Style Web Service, you must include the <code>&lt;message-style&gt;</code> tag and specify the value <code>doc</code>.</p> <p>Valid Values: <code>doc</code>, <code>rpc</code></p> <p>This tag is required for Document Style Web Services.</p> <p>Default value: <code>rpc</code> (when the <code>&lt;message-style&gt;</code> tag is not supplied)</p>
<pre>&lt;scope&gt; value &lt;/scope&gt;</pre>	<p>The <code>&lt;scope&gt;</code> tag only applies for stateful services. Use this tag only within the <code>&lt;stateful-java-service&gt;</code> tag.</p> <p>This tag is optional.</p> <p>Valid Values: <code>application</code>, <code>session</code></p> <p>Default Value: <code>session</code></p>
<pre>&lt;session-timeout&gt; value &lt;/session-timeout&gt;</pre>	<p>This optional parameter only applies for stateful services. Use this tag only within the <code>&lt;stateful-java-service&gt;</code> tag.</p> <p>Specify <i>value</i> with an integer that defines the timeout for the session timeout. The default value for the session timeout for stateful Java sessions where no session timeout is specified is 60 seconds.</p> <p>This tag is optional.</p>
<pre>&lt;uri&gt; URI &lt;/uri&gt;</pre>	<p>This tag specifies servlet mapping pattern for the Servlet that implements the Document Style Web Service. The path specified as the <i>URI</i> is appended to the <code>&lt;context&gt;</code> to specify the Document Style Web Service location.</p> <p>This tag is optional.</p>

### Adding Stateful Document Style Java Service Tags

[Table 6–3](#) describes the `<stateful-java-service>` `WebServicesAssembler` configuration file tags. Use these tags when creating a configuration file for a stateful Document Style Web Service.

[Example 6–5](#) shows a complete `config.xml` file, including the stateful Document Style Web Service tags.

### Adding Stateless Document Style Java Service Tags

[Table 6–3](#) describes the `<stateless-java-service>` `WebServicesAssembler` configuration file tags to use when creating a stateful Document Style Web Service. The `<stateless-java-service>` tag is included within a `<web-service>` tag in the configuration file. Add this tag to provide information required for generating a stateless Document Style Web Service J2EE .ear file.

[Example 6–6](#) shows a complete `config.xml` file, including the stateless Document Style Web Service tags.

---

**Note:** Deploying a stateful Java implementation class as a stateless Document Style Web Service could yield unpredictable results.

---

### Adding WSDL and Client-Side Proxy Generation Tags

The `WebServicesAssembler` configuration file supports the `<wsdl-gen>` and `<proxy-gen>` tags to allow a Web Service developer to generate Web Service description WSDL files and client-side proxy files. You can add these tags to control

whether the WSDL file and the client-side proxy are generated. You can also specify that the WSDL file be assembled with the Document Style Web Service .ear. A client-side developer can then obtain the WSDL file from the deployed Web Service and use it to build an application.

**See Also:** ["Generating WSDL Files and Client Side Proxies"](#) on page 9-3

**Example 6–5 Sample Stateful Java WebServicesAssembler Configuration File for a Document Style Web Service**

```
<web-service>
  <display-name>Stateful Java Document Web Service</display-name>
  <description>Stateful Java Document Web Service Example</description>
  <!-- Specifies the resulting web service archive will be stored in ./docws.ear -->
  <destination-path>./docws.ear</destination-path>
  <!-- Specifies the temporary directory that web service assembly tool can create temporary files. -->
  <temporary-directory>./temp</temporary-directory>
  <!-- Specifies the web service will be accessed in the servlet context named "/docws". -->
  <context>/statefuldocws</context>

  <!-- Specifies the web service will be stateful -->

  <stateful-java-service>
    <interface-name>StatefulDoc</interface-name>
    <class-name>StatefulDocImpl</class-name>
    <!-- Specifies the web service will be accessed in the uri named "/docService" within the servlet
context. -->
    <uri>/docservice</uri>
    <!-- Specifies the location of Java class files ./classes -->
    <java-resource>./classes</java-resource>
    <!-- Specifies that it uses document style SOAP messaging -->
    <message-style>doc</message-style>
  </stateful-java-service>

  <!-- generate the wsdl -->
  <wsdl-gen>
<wsdl-dir>wsdl</wsdl-dir>
  <!-- over-write a pregenerated wsdl , turn it 'false' to use the pregenerated wsdl-->
  <option name="force">true</option>
  <option name="httpServerURL">http://localhost:8888</option>
  </wsdl-gen>

  <!-- generate the proxy -->

  <proxy-gen>
<proxy-dir>proxy</proxy-dir>
  <option name="include-source">true</option>
  </proxy-gen>
</web-service>
```

**Example 6–6 Sample Stateless Java WebServicesAssembler Configuration File for a Document Style Web Service**

```
<web-service>
  <display-name>Stateless Java Document Web Service</display-name>
  <description>Stateless Java Document Web Service Example</description>
  <!-- Specifies the resulting web service archive will be stored in ./statelessdocws.ear -->
  <destination-path>./statelessdocws.ear</destination-path>
  <!-- Specifies the temporary directory that web service assembly tool can create temporary files. -->
  <temporary-directory>./temp</temporary-directory>
```



```

<!-- Specifies the web service will be accessed in the servlet context named "/statelessdocws". -->
<context>/statelessdocws</context>
<!-- to package the stylesheet to format input xml -->
<option name="source-path">converter.xml</option>

<!-- Specifies the web service will be stateless -->

<stateless-java-service>
  <interface-name>StatelessDoc</interface-name>
  <class-name>StatelessDocImpl</class-name>
  <!-- Specifies the web service will be accessed in the uri named "/docService" within the servlet
context. -->
  <uri>/docservice</uri>
  <!-- Specifies the location of Java class files ./classes -->
  <java-resource>./classes</java-resource>
  <!-- Specifies that it uses document style SOAP messaging -->
  <message-style>doc</message-style>
</stateless-java-service>

<!-- generate the wsdl -->
<wsdl-gen>
<wsdl-dir>wsdl</wsdl-dir>
<!-- over-write a pregenerated wsdl , turn it 'false' to use the pregenerated wsdl-->
<option name="force">true</option>
<option name="httpServerURL">http://localhost:8888</option>
</wsdl-gen>

<!-- generate the proxy -->
<proxy-gen>
<proxy-dir>proxy</proxy-dir>
<option name="include-source">true</option>
</proxy-gen>

</web-service>

```

## Running WebServicesAssembler With Document Style Web Services

After you create the `WebServicesAssembler` configuration file, you can generate a J2EE .ear file for the Document Style Web Service. The J2EE EAR file includes Document Style Web Service servlet configuration information, including the generated file `web.xml`, and the implementation classes.

Run the Oracle Application Server Web Services assembly tool, `WebServicesAssembler` as follows:

```
java -jar WebServicesAssembler.jar -config my_service_config
```

Where: *my\_service\_config* is the configuration file that contains the `<stateless-java-service>` or the `<stateful-java-service>` tag.

### See Also:

- ["Creating a Configuration File to Assemble Document Style Web Services"](#) on page 6-7
- ["Running the Web Services Assembly Tool"](#) on page 9-1

## Deploying Document Style Web Services

After creating the .ear file containing Java classes and the Web Services Servlet deployment descriptors, you can deploy the Web Service as you would any standard J2EE application stored in an .ear file (to run under OC4J).

**See Also:** *Oracle Application Server Containers for J2EE User's Guide*  
in the Oracle Application Server 10g Documentation Library

---

# Developing and Deploying JMS Web Services

This chapter describes the procedures you use to configure, deploy, and build Oracle Application Server Web Services that expose JMS destinations, including JMS Queues and JMS Topics as Web Services. This chapter also covers writing a backend JMS message processor to consume incoming JMS messages and to generate outgoing JMS messages.

Oracle Application Server Web Services supports asynchronous message facilities with JMS Web Services.

This chapter covers the following topics:

- [JMS Web Services Overview](#)
- [Writing JMS Web Services and Handling Messages](#)
- [Preparing and Configuring JMS Web Services](#)
- [Deploying JMS Web Services](#)
- [Limitations for JMS Web Services](#)

## JMS Web Services Overview

This section covers the following topics:

- [Using JMS Web Services](#)
- [JMS Web Services Backend Message Processing](#)

## Using JMS Web Services

The sample code for JMS Web Services is supplied on the Oracle Technology Network Web site,

<http://otn.oracle.com/tech/java/oc4j/demos/1012/index.html>

After expanding the Web Services `demo.zip` file, the samples are in the `demo1` and `demo2` directories under `webservices/demo/basic/jms_service` on UNIX and `webservices\demo\basic\jms_service`.

JMS Web Services examples show both OC4J/JMS and Oracle JMS. In the samples, `demo1` uses OC4J/JMS and `demo2` uses Oracle JMS.

Using JMS Web Services, Oracle Application Server supplies a Servlet that supports two operations on messages: a `send` operation and a `receive` operation. Using these two operations, if the destination is a JMS Queue, `send` means `enqueue`, and `receive`

means dequeue. If the destination is a topic, `send` means publish and `receive` means subscribe. An individual JMS Web Service can support just the `send` operation, just the `receive` operation, or both operations, as determined by the service developer.

The JMS Web Service determines how to handle incoming and outgoing messages for JMS destinations based on the configuration of the JMS Web Service and on the operation specified by the client-side program that uses the JMS Web Service. The Oracle Application Server Web Services runtime verifier throws an exception if the operation supplied by a JMS Web Service client is invalid. For example, if the deployment operation is `send`, and the request is `receive`, an exception is thrown.

The client-side message associated with a JMS Web Service is an XML document that conforms to the Document Object Model (DOM) as specified by the W3C (`org.w3c.dom.Element`). For a `send` operation, it is the client-side developer's job to deliver a message of the correct form to a JMS Web Service. And likewise, for a `receive` operation, the client must handle the message it receives from a JMS Web Service.

**See Also:** <http://java.sun.com/products/jms/> for information on JMS

## JMS Web Services Backend Message Processing

A JMS Web Service consists of configuration information that defines the Web Service, and, in addition the server-side developer provides code that consumes the messages that a JMS Web Service client sends, or generates the messages that the client receives.

This section describes the architecture for processing JMS messages associated with a JMS Web Service and covers the following topics:

- [Using an MDB for Message Processing](#)
- [Using a JMS Client for Message Processing](#)

### Using an MDB for Message Processing

A JMS Web Service either sends messages to a JMS destination or receives messages from a JMS destination and can use an MDB on the backend for generating and consuming messages. For example, [Figure 7-1](#) shows an MDB based JMS Web Service that, from the JMS Web Service client's view, handles both the message `send` and the message `receive` operations.

Figure 7-1 MDB Based JMS Web Service

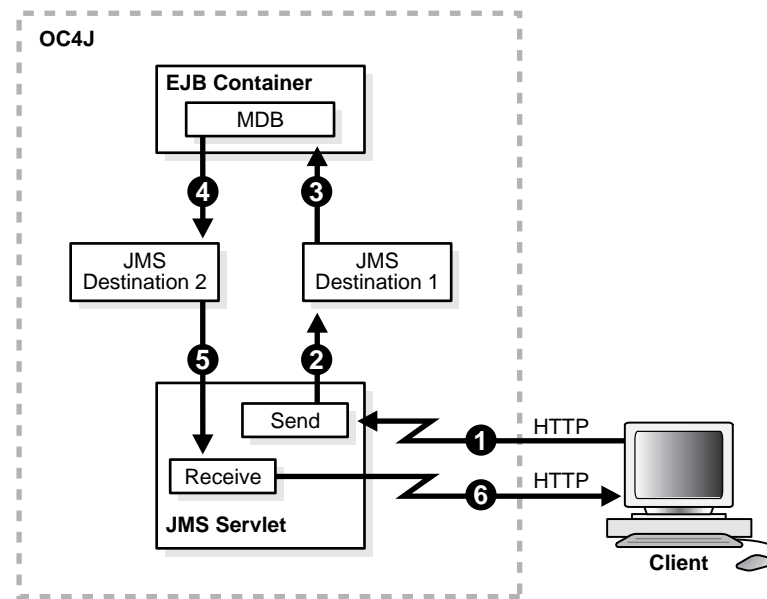


Figure 7-1 includes an MDB that is configured to listen to a JMS destination. The MDB based JMS Web Service works with the following steps:

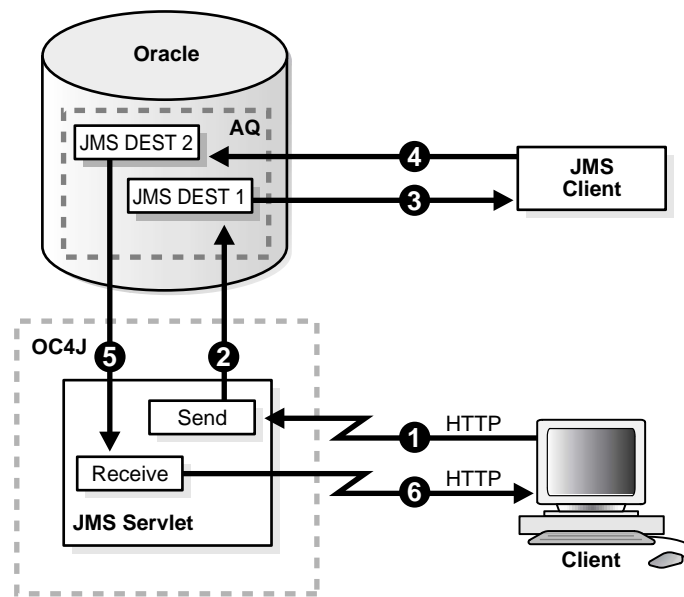
1. A JMS Web Service client performs a send operation on the JMS Web Service to send a message.
2. The JMS Web Service processes the incoming message and directs it to a JMS destination, JMS Destination 1.
3. The EJB container invokes the MDB listening on JMS Destination 1.
4. After processing the message an MDB produces a new message on JMS Destination 2. Producing and consuming messages could involve one or more MDBs. For example, a single MDB could be listening on JMS Destination 1 and the same MDB could also send the message to JMS Destination 2.
5. (Arrows 5 and 6) A JMS Web Service client performs a receive operation on the JMS Web Service to receive a message. The JMS Web Service consumes a message from the JMS destination, processes it, and passes the outgoing message to the client.

### Using a JMS Client for Message Processing

Using a JMS client for message processing, the JMS Web Service does not assemble, deploy, or run the JMS code on the backend. A separate JMS program that runs outside of the JMS Web Service, as a standalone JMS client, is responsible for generating and consuming the JMS messages that are associated with the JMS Web Service.

For example, Figure 7-2 shows a JMS Web Service that use a server-side JMS client for message processing.

Figure 7-2 JMS Client Based JMS Web Service



The JMS Web service includes only configuration information that supports handling messages and using JMS destinations. The JMS client based JMS Web Service works with the following steps:

1. A JMS Web Service client performs a send operation on the JMS Web Service to send a message.
2. The JMS Web Service then processes the incoming message and directs it to JMS DEST 1.
3. The JMS client processes the incoming message on JMS DEST 1. The incoming message could be identified using a message listener, or by other means.
4. After processing the incoming message the JMS client may produce a new message on JMS DEST 2. The message on JMS DEST 2 could be produced by another JMS client or by the same JMS client.
5. (Arrows 5 and 6) A JMS Web Service client performs a receive operation on the JMS Web Service to receive a message. The JMS Web Service consumes an outgoing message from the JMS destination and passes the message to the client.

## Writing JMS Web Services and Handling Messages

Writing a JMS Web Service presents a server-side developer with two tasks:

1. Building the backend message processing program for a JMS Web Service.
2. Preparing and configuring a JMS Web Service.

This section covers the following:

- [Using an MDB for Backend Message Processing](#)
- [Using a JMS Standalone Program for Backend Message Processing](#)
- [Message Processing and Reply Messages](#)

**See Also:**

- ["Preparing and Configuring JMS Web Services"](#) on page 7-8
- [Chapter 4, "Developing and Deploying EJB Web Services"](#)

## Using an MDB for Backend Message Processing

When a JMS Web Service uses an MDB for generating or consuming messages, the MDB must be assembled with the JMS Web Service. In this case, the MDB is packaged as part of the J2EE .ear file that is deployed as a JMS Web Service.

Using an MDB with a JMS Web Service, the server-side developer is responsible for performing the following steps:

- [Developing the MDB that Processes Incoming Messages](#)
- [Developing the MDB that Generates Outgoing Messages](#)
- [Compiling and Preparing the MDB EJB.jar File](#)
- [Assembling the JMS Web Service With the MDB](#)
- [Defining the Server-Side Resource References](#)

---



---

**Note:** A given JMS Web Service may process incoming messages, generate outgoing messages, or do both.

---



---

### Developing the MDB that Processes Incoming Messages

The MDB that processes incoming messages, generated from a JMS Web Service send operation, must include an `onMessage()` method with the following characteristics:

- The `onMessage()` method should be declared as `public`, but not `final` or `static`
- The `onMessage()` method should have a return type of `void`
- The `onMessage()` method should have one argument of type `javax.jms.Message`. The JMS Web Service only supports messages of type `ObjectMessage`, so the MDB developer should cast the incoming JMS Web Service message to an `ObjectMessage`.
- The message payload is available from the message using the `getObject()` method on the incoming JMS message and casting to the `Element` type.

[Example 7-1](#) shows an MDB method that handles an incoming JMS Message. Also see `MessageBean.java` in the `demo1` directory for the complete code.

#### **Example 7-1** Sample Incoming `onMessage()` Method for JMS Web Service

```
public void onMessage(Message inMessage) {
    ObjectMessage msg = null;
    Element e;
    try {
        // Message should be of type objectMessage
        if (inMessage instanceof ObjectMessage) {
            // retrieve the object
            msg = (ObjectMessage) inMessage;
            e = (Element)msg.getObject();
            processElement(e);
            this.send2Queue(e);
        } else {
```

```

        System.out.println("MessageBean::onMessage() => Message of wrong type: "
+ inMessage.getClass().getName());
    }
    } catch (JMSException ex) {
        ex.printStackTrace();
        mdc.setRollbackOnly();
    } catch (Throwable te) {
        te.printStackTrace();
    }
}
}

```

## Developing the MDB that Generates Outgoing Messages

An MDB that generates an outgoing message, consumed by a JMS Web Service receive operation, must include code that produces a message on a JMS destination with the following characteristics:

- The message placed on the JMS destination should be of type: `javax.jms.Message.ObjectMessage`.
- Set the payload of the message using the `setObject()` method on the outgoing JMS message and casting to the `java.io.Serializable` type.

[Example 7-2](#) shows a code fragment that creates an outgoing message of the correct type. For the complete code for this example, see `MessageBean2.java` in the `demo2` directory.

### **Example 7-2** Sample Outgoing Message for JMS Web Service

```

// Create an Object Message
message = queueSession.createObjectMessage();
// Stuff the result into the ObjectMessage
((ObjectMessage)message).setObject ((java.io.Serializable)ee);
// Send the Message
queueSender.send(message);

```

## Compiling and Preparing the MDB EJB.jar File

After compiling the MDB classes, create an EJB .jar file that includes the MDB and its required deployment information.

## Assembling the JMS Web Service With the MDB

Assemble the MDB's EJB.jar file with the JMS Web Service .ear file using the `WebServicesAssembler` tool and a configuration file containing the top-level tag `<option name=source-path">` that specifies the EJB .jar, and the `<jms-doc-service>` that defines the JMS Web Service configuration.

### **See Also:**

- ["Preparing and Configuring JMS Web Services"](#) on page 7-8
- ["Deploying JMS Web Services"](#) on page 7-13



## Defining the Server-Side Resource References

Define the resource references associated with the JMS destinations that the JMS Web Service uses:

- If the MDB uses OC4J/JMS, define the resource references in the OC4J `jms.xml` configuration file.
- If the MDB uses Oracle JMS, then run the `sql` files that support access to the Oracle JMS destinations.

**See Also:** Chapter 3, "AQ Programmatic Environments" in the *Application Developer's Guide - Advanced Queuing* in the Oracle9i Database Documentation library

## Using a JMS Standalone Program for Backend Message Processing

Using a JMS standalone program on the backend for the JMS Web Service, the server-side developer is responsible for performing the following steps:

1. Developing the JMS client that defines the JMS destinations, handles incoming messages, processes them, and produces the outgoing messages. The JMS client can also perform processing that uses a JMS destination that triggers an MDB.
2. Assembling the JMS Web Service `.ear` file using the `WebServicesAssembler` tool and a configuration file containing the top-level tag `<jms-doc-service>`.
3. Defining the resource references associated with JMS destinations in the OC4J/JMS `jms.xml` configuration file. If the JMS destinations are defined in Oracle JMS, then the developer must run the `sql` files that initialize the access to the Oracle JMS destinations.

**See Also:**

- ["Using an MDB for Backend Message Processing"](#) on page 7-5
- ["Deploying JMS Web Services"](#) on page 7-13

---



---

**Note:** When a JMS Web Service uses standalone a JMS client to consume or generate messages, the standalone client cannot be assembled with the JMS Web Service.

---



---

## Message Processing and Reply Messages

The JMS Web Service processes an incoming message, a JMS Web Service `send` operation message, and places the message on a JMS destination. This section covers details that a developer needs to know to consume and process the JMS messages that originate from a JMS Web Service.

The client-side message associated with a JMS Web Service is an XML document that conforms to the Document Object Model (DOM) as specified by the W3C (`org.w3c.dom.Element`). When a JMS Web Service is sent an `Element` from a Web Service client, it creates a `JMS ObjectMessage` that contains the `Element`. The JMS Web Service may set certain header values before it places the message on a JMS destination. Depending on the values of optional configuration tags specified when the JMS Web Service is assembled, the JMS Web Service sets the following JMS Message Headers:

`JMSType`  
`JMSReplyTo`

```
JMSExpiration
JMSPriority
JMSDeliveryMode
```

When the JMS Web Service sets the `JMSReplyTo` header, it uses either the value specified with the `<reply-to-topic-resource-ref>` or the `<reply-to-queue-resource-ref>` (only one of these should be configured for any given JMS Web Service). The value specified with the `<reply-to-connection-factory-resource-ref>` tag is set on the message as a standard string property. The property name is `OC4J_REPLY_TO_FACTORY_NAME`.

[Example 7-3](#) provides a code segment that shows where the `onMessage()` method gets the `ReplyTo` information for message generated from a JMS Web Service send operation:

**Example 7-3**

```
public void onMessage(Message inMessage) {
    // Do some processing
    ObjectMessage msg = null;
    String      factoryName;
    Destination dest;
    Element     el;
    try {
        // Message should be of type objectMessage
        if (inMessage instanceof ObjectMessage) {
            // retrieve the object
            msg = (ObjectMessage) inMessage;
            el = (Element)msg.getObject();
            System.out.println("MessageBean2::onMessage() => Message received: " );
            ((XMLElement)el).print(System.out);
            processElement(el);
            factoryName = inMessage.getStringProperty("OC4J_REPLY_TO_FACTORY_NAME");
            dest = inMessage.getJMSReplyTo();
        }
    }
}
```

**See Also:**

- ["Developing the MDB that Processes Incoming Messages"](#) on page 7-5
- ["Adding JMS Doc Service Tags"](#) on page 7-10

## Preparing and Configuring JMS Web Services

This section describes how to use the Oracle Application Server Web Services tool `WebServicesAssembler` to prepare a J2EE .ear file for a JMS Web Service.

To deploy a JMS Web Service, you need to assemble a J2EE .ear file. The J2EE .ear file can include the following:

- The deployment descriptors for the Oracle Application Server Web Services Servlet.
- If the JMS Web Service also includes an MDB, then the J2EE .ear also includes a Jar file that supplies the MDB implementation. This component is optional. To expose JMS Queues or Topics as JMS Web Services, you are not required to include an MDB Jar file with the JMS Web Service.

This section describes the procedures you use to create a configuration file to use with the `WebServicesAssembler`.

This section contains the following topics:

- [Creating a Configuration File to Assemble JMS Web Services](#)
- [Running WebServicesAssembler With JMS Web Services](#)

## Creating a Configuration File to Assemble JMS Web Services

The Oracle Application Server Web Services assembly tool, `WebServicesAssembler`, assists in assembling Oracle Application Server Web Services. This section describes how to create an XML configuration file that describes the JMS Web Service to be assembled.

Create `WebServicesAssembler` configuration file by adding the following:

- [Adding Web Service Top Level Tags](#)
- [Adding JMS Doc Service Tags](#)
- [Adding WSDL and Client-Side Proxy Generation Tags](#)

### Adding Web Service Top Level Tags

[Table 7-1](#) describes the top level `WebServicesAssembler` configuration file tags. Add these tags to provide top level information describing the JMS Web Service.

[Example 7-4](#) shows a complete JMS Web Service sample configuration file. The `demo1` and `demo2` directories in the `jms_service` directory contain complete `config.xml` files for JMS Web Services.

**Table 7-1 Top Level `WebServicesAssembler` Configuration Tags**

Tag	Description
<code>&lt;context&gt;</code> <i>context</i> <code>&lt;/context&gt;</code>	Specifies the context root of the Web Service. This tag is required.
<code>&lt;datasource-JNDI-name&gt;</code> <i>name</i> <code>&lt;/datasource-JNDI-name&gt;</code>	Specifies the datasource associated with the Web Service.
<code>&lt;description&gt;</code> <i>description</i> <code>&lt;/description&gt;</code>	Provides a simple description of the Web Service. This tag is optional.
<code>&lt;destination-path&gt;</code> <i>dest_path</i> <code>&lt;/destination-path&gt;</code>	Specifies the name of the generated J2EE .ear file output. The <i>dest_path</i> specifies the complete path for the output file. This tag is required.
<code>&lt;display-name&gt;</code> <i>disp_name</i> <code>&lt;/display-name&gt;</code>	Specifies the Web Service display name. This tag is optional.

**Table 7-1 (Cont.) Top Level WebServicesAssembler Configuration Tags**

Tag	Description
<code>&lt;option name="source-path"&gt;</code> <i>path</i> <code>&lt;/option&gt;</code>	<p>Includes a specified file in the output .ear file. Use this option to specify java resources, or the name of an existing .war, .ear, or ejb-jar file that is used as a source file for the output J2EE .ear file.</p> <p>When a .war file is supplied as input, the optional contextroot specifies the root-context for the .war file.</p> <p><i>path1</i> specifies the context-root for the .war.</p> <p><i>path2</i> specifies the path to the file to include.</p> <p>For example:</p> <pre>&lt;option name="source-path" contextroot="/test"&gt;/myTestArea/ws/src/statefull.war&lt;/option&gt;</pre> <p>This tag is optional.</p>
<code>&lt;jms-doc-service&gt;</code> <i>sub-tags</i> <code>&lt;/jms-doc-service&gt;</code>	<p>Use this tag to add a JMS Web Service. See <a href="#">Table 7-2</a> for a description of the valid <i>sub-tags</i>.</p>
<code>&lt;temporary-directory&gt;</code> <i>temp_dir</i> <code>&lt;/temporary-directory&gt;</code>	<p>Specifies a directory where the assembler can store temporary files.</p> <p>This tag is optional.</p>

### Adding JMS Doc Service Tags

The `<jms-doc-service>` defines the configuration information for a JMS Web Service. The JMS Web Service developer determines if the service supports send operations, receive operations, or both send and receive, based on the value of the `<operation>` sub-tag. Some of the configuration file tags are only valid, depending on the operation selected for the Web Service. [Table 7-2](#) lists all the supported `<jms-doc-service>` sub-tags, and includes information on whether each is valid, based on the operation specified.

**Table 7-2 JMS Service WebServicesAssembler Configuration Tags**

Tag	Description
<code>&lt;connection-factory-resource-ref&gt;</code> <i>resource-ref</i> <code>&lt;/connection-factory-resource-ref&gt;</code>	<p>Specifies the Topic Connection Factory or Queue Connection Factory resource reference <i>resource-ref</i> for the JMS destination associated with the JMS Web Service.</p> <p>This tag is required.</p>
<code>&lt;jms-delivery-mode&gt;</code> <i>delivery-mode</i> <code>&lt;/jms-delivery-mode&gt;</code>	<p>Sets the JMSDeliveryMode message header to the specified <i>delivery-mode</i> value for the JMS message that is created with a send operation.</p> <p>This tag is valid when the <code>&lt;operation&gt;</code> value is: send or both</p> <p>This tag is optional.</p>
<code>&lt;jms-expiration&gt;</code> <i>expiration</i> <code>&lt;/jms-expiration&gt;</code>	<p>Sets the JMSEExpiration message header to the specified <i>expiration</i> value for the JMS message that is created with a send operation.</p> <p>This tag is valid when the <code>&lt;operation&gt;</code> value is: send or both</p> <p>This tag is optional.</p>
<code>&lt;jms-message-type&gt;</code> <i>message-type</i> <code>&lt;/jms-message-type&gt;</code>	<p>Sets the JMSType for the message to the specified <i>message-type</i> for the JMS message that is created with a send operation</p> <p>This tag is valid when the <code>&lt;operation&gt;</code> value is: send or both</p> <p>This tag is optional.</p>

**Table 7-2 (Cont.) JMS Service WebServicesAssembler Configuration Tags**

Tag	Description
<pre>&lt;jms-priority&gt; priority &lt;/jms-priority&gt;</pre>	<p>Sets the <code>JMSPriority</code> message header to the specified <i>priority</i> value for the JMS message that is created with a send operation.</p> <p>This tag is valid when the <code>&lt;operation&gt;</code> value is: send or both</p> <p>This tag is optional.</p>
<pre>&lt;receive-timeout&gt; timeout &lt;/receive-timeout&gt;</pre>	<p>Provides a configurable timeout value to specify the receive timeout in milliseconds. This specifies the time in milliseconds that a receive operation waits for a new message.</p> <p>This tag is valid when the <code>&lt;operation&gt;</code> value is: receive or both</p> <p>When this tag is not specified or when the value is set to 0, a JMS receive operation blocks indefinitely. Valid values are 0 and positive integers.</p> <p>Default value: 0</p> <p>This tag is optional.</p>
<pre>&lt;operation&gt; op &lt;/operation&gt;</pre>	<p>Specifies the operation <i>op</i> that the JMS Web Service supports.</p> <p>Using the send and receive operation:</p> <ul style="list-style-type: none"> <li>■ If the destination is a JMS Queue, send means enqueue, and receive means dequeue.</li> <li>■ If the destination is a topic, send means publish and receive means subscribe.</li> </ul> <p>The send operation uses the <code>&lt;connection-factory-resource-ref&gt;</code> and the corresponding JMS destination <code>&lt;queue-resource-ref&gt;</code> or <code>&lt;topic-resource-ref&gt;</code> to determine the JMS destination for a send operation on the service.</p> <p>With the receive operation, when the <code>&lt;reply-to-connection-factory-resource-ref&gt;</code> tag is not set, then the receive operation uses the <code>&lt;connection-factory-resource-ref&gt;</code> and the corresponding JMS destination <code>&lt;queue-resource-ref&gt;</code> or <code>&lt;topic-resource-ref&gt;</code>. When the <code>&lt;reply-to-connection-factory-resource-ref&gt;</code> tag is set, then the <code>&lt;reply-to-*&gt;</code> tags specify the JMS destination for receive operations.</p> <p>Valid values: send, receive, both</p> <p>Default value: both</p> <p>This tag is optional.</p>
<pre>&lt;queue-resource-ref&gt; queue-ref &lt;/queue-resource-ref&gt;</pre>	<p>Specifies the resource reference <i>queue-ref</i> of the destination JMS queue.</p> <p>Either a <code>&lt;topic-resource-ref&gt;</code> or a <code>&lt;queue-resource-ref&gt;</code> must be specified, but not both. When a <code>&lt;queue-resource-ref&gt;</code> is specified, the <code>&lt;connection-factory-resource-ref&gt;</code> must refer to a corresponding Queue connection factory.</p>
<pre>&lt;reply-to-connection-factory-resource-ref&gt; reply-to-conn-factory-res-ref &lt;/reply-to-connection-factory-resource-ref&gt;</pre>	<p>If the <code>&lt;operation&gt;</code> specified is both, then receive operations use the <code>&lt;reply-to-connection-factory-resource-ref&gt;</code>. The specified <i>reply-to-conn-factory-res-ref</i> value specifies the JMS destination connection factory for receive operations. Also, if the MDB, or any JMS consumer, expects to send results back then the name of the destination connection factory to which the reply message will be sent has to be specified in this parameter.</p> <p>See Also: <a href="#">"Message Processing and Reply Messages"</a> on page 7-7.</p> <p>This tag is optional.</p>

**Table 7–2 (Cont.) JMS Service WebServicesAssembler Configuration Tags**

Tag	Description
<code>&lt;reply-to-queue-resource-ref&gt;</code> <i>reply-to-queue-res-ref</i> <code>&lt;/reply-to-queue-resource-ref&gt;</code>	<p>Specifies the resource reference <i>reply-to-queue-res-ref</i> of the destination JMS queue.</p> <p>When a <code>&lt;reply-to-queue-resource-ref&gt;</code> is specified, the <code>&lt;reply-to-connection-factory-resource-ref&gt;</code> must refer to a corresponding Queue connection factory.</p> <p>If the <code>&lt;reply-to-connection-factory-resource-ref&gt;</code> tag is set, then either a <code>&lt;reply-to-topic-resource-ref&gt;</code> or a <code>&lt;reply-to-queue-resource-ref&gt;</code> must be specified, but not both.</p> <p>This tag is optional.</p>
<code>&lt;reply-to-topic-resource-ref&gt;</code> <i>reply-to-topic-res-ref</i> <code>&lt;/reply-to-topic-resource-ref&gt;</code>	<p>Specifies the resource reference <i>reply-to-topic-res-ref</i> of the destination JMS Topic.</p> <p>When a <code>&lt;reply-to-topic-resource-ref&gt;</code> is specified, the <code>&lt;reply-to-connection-factory-resource-ref&gt;</code> must refer to a corresponding Topic connection factory.</p> <p>If the <code>&lt;reply-to-connection-factory-resource-ref&gt;</code> tag is set, then either a <code>&lt;reply-to-topic-resource-ref&gt;</code> or a <code>&lt;reply-to-queue-resource-ref&gt;</code> must be specified, but not both.</p> <p>This tag is optional.</p>
<code>&lt;topic-resource-ref&gt;</code> <i>topic-ref</i> <code>&lt;/topic-resource-ref&gt;</code>	<p>Specifies the resource reference <i>topic-ref</i> of the destination JMS Topic.</p> <p>Either a <code>&lt;topic-resource-ref&gt;</code> or a <code>&lt;queue-resource-ref&gt;</code> must be specified, but not both. When a <code>&lt;topic-resource-ref&gt;</code> is specified, the <code>&lt;connection-factory-resource-ref&gt;</code> must refer to a corresponding Topic connection factory.</p>
<code>&lt;topic-subscription-name&gt;</code> <i>topic-name</i> <code>&lt;/topic-subscription-name&gt;</code>	<p>When a JMS provider supports durable JMS topics, the JMS Doc service supports using the durable topics. To specify a durable topic, use this tag to specify the <i>topic-name</i>. This tag is only valid when a <code>&lt;topic-resource-ref&gt;</code> is supplied.</p> <p>This tag is optional.</p>
<code>&lt;uri&gt;</code> <i>URI</i> <code>&lt;/uri&gt;</code>	<p>This tag specifies servlet mapping pattern for the Servlet that implements the JMS Web Service. The path specified as the <i>URI</i> is appended to the <code>&lt;context&gt;</code> to specify the JMS Web Service location.</p> <p>This tag is optional.</p>

### Adding WSDL and Client-Side Proxy Generation Tags

The `WebServicesAssembler` supports the `<wsdl-gen>` and `<proxy-gen>` tags to allow a Web Service developer to generate WSDL files and client-side proxy files. You can use these tags to control whether the WSDL file and the client-side proxy are generated. Using these tags you can also specify that the generated WSDL file or a WSDL file that you write is packaged with the Web Service J2EE .ear.

A client-side developer either uses the WSDL file that is obtained from a deployed Web Service, or the client-side proxy that is generated from the WSDL to build an application that uses the Web Service.

**See Also:** ["Generating WSDL Files and Client Side Proxies"](#) on page 9-3

#### Example 7–4 Sample WebServicesAssembler Configuration File for JMS Web Service

```

<web-service>
  <display-name>JMS Web Service Example</display-name>
  <description>JMS Web Service Example</description>
  <!-- Name of the destination -->
  <destination-path>./jmsws1.ear</destination-path>

```

```

<temporary-directory>./tmp</temporary-directory>
<!-- Context root of the application -->
<context>/jmswsl</context>
<!-- Path of the jar file with MDBs defined/implemented in it -->
<option name="source-path">MDB/mdb_servicel.jar</option>

<!-- tags for jms doc service -->
<jms-doc-service>
  <uri>JmsSend</uri>
  <connection-factory-resource-ref>jms/theQueueConnectionFactory</connection-factory-resource-ref>
  <queue-resource-ref>jms/theQueue</queue-resource-ref>
  <operation>send</operation>x
</jms-doc-service>

<jms-doc-service>
  <uri>JmsReceive</uri>
  <connection-factory-resource-ref>jms/logQueueConnectionFactory</connection-factory-resource-ref>
  <queue-resource-ref>jms/logQueue</queue-resource-ref>
  <operation>receive</operation>
</jms-doc-service>
<!-- generate the wsdl -->
<wsdl-gen>
  <wsdl-dir>wsdl</wsdl-dir>
  <!-- over-write a pregenerated wsdl , turn it 'false' to use the pregenerated wsdl-->
  <option name="force">>true</option>
  <option name="httpServerURL">http://localhost:8888</option>
  <!-- do not package the wsdl -generate it again on the server-->
  <option name="packageIt">>false</option>
</wsdl-gen>
<!-- generate the proxy -->
<proxy-gen>
  <proxy-dir>proxy</proxy-dir>
  <option name="include-source">>true</option>
</proxy-gen>
</web-service>

```

## Running WebServicesAssembler With JMS Web Services

After you create the `WebServicesAssembler` configuration file, you can generate a J2EE .ear file for the JMS Web Service. The J2EE EAR file includes Web Service servlet configuration information, including the generated file `web.xml`, and if the service includes MDBs, the `ejb.jar` file containing the implementation classes.

Run the Oracle Application Server Web Services assembly tool, `WebServicesAssembler` as follows:

```
java -jar WebServicesAssembler.jar -config my_jms_service_config
```

Where: `my_jms_service_config` is the configuration file that contains the `<jms-doc-service>` tag.

### See Also:

- ["Creating a Configuration File to Assemble JMS Web Services"](#) on page 7-9
- ["Running the Web Services Assembly Tool"](#) on page 9-1

## Deploying JMS Web Services

After creating the .ear file containing Java classes and the Web Services Servlet deployment descriptors, you can deploy the Web Service as you would any standard J2EE application stored in an .ear file (to run under OC4J).

**See Also:** *Oracle Application Server Containers for J2EE User's Guide*  
in the Oracle Application Server 10g Documentation Library

## Limitations for JMS Web Services

The JMS Web Service only supports messages of type `ObjectMessage`  
(`javax.jms.Message.ObjectMessage`).



---

# Building Clients that Use Web Services

This chapter describes the Oracle Application Server Web Services features that allow you to easily create and run a client application that uses Oracle Application Server Web Services.

This chapter contains the following topics:

- [Locating Web Services](#)
- [Getting WSDL Files and Client-Side Proxy Jars for Web Services](#)
- [Working with Client-Side Proxy Jar to Use Web Services](#)
- [Working with WSDL Files and Oracle JDeveloper to Use Web Services](#)

## Locating Web Services

When you want to use Web Services you need to develop a client application. There are two types of Web Services clients: static web service clients and dynamic web service clients. A **static web service client** knows where a Web Service is located without looking up the service in a UDDI registry. A **dynamic web service client** performs a lookup to find the Web Service's location in a UDDI registry before accessing the service. [Chapter 10, "Discovering and Publishing Web Services"](#) provides detailed information on looking up Web Services in a UDDI registry.

Using a static client Oracle Application Server Web Services provides several options for locating Oracle Application Server Web Services, including:

- Using a known Web Service located at a known URL.
- Using Oracle Application Server Web Services and a known service URL to obtain a client-side proxy Jar, or by other means obtaining a client-side proxy Jar for a Web Service. The client-side proxy Jar that Oracle Application Server Web Services generates includes the URL to locate the associated Web Service.
- Using Oracle Application Server Web Services and a known service URL to obtain a WSDL file, or by other means obtaining a WSDL file that describes a Web Service. The WSDL files that Oracle Application Server Web Services generates includes the URL to locate the associated Web Service.

After you locate a Web Service or after you obtain either the WSDL or client-side proxy Jar, you can build a client-side application that uses the Web Service.

**See Also:** [Chapter 10, "Discovering and Publishing Web Services"](#)

## Getting WSDL Files and Client-Side Proxy Jars for Web Services

This section covers the following:

- [Using the Web Service Home Page to Save WSDL and Client Side Proxies](#)
- [Getting Web Service WSDL and Client-Side Proxies Directly](#)
- [Generating Client-Side Proxies With WebServicesAssembler](#)

### Using the Web Service Home Page to Save WSDL and Client Side Proxies

To use Oracle Application Server Web Services you need to create a client-side application that accesses a Web Service. Oracle Application Server Web Services supplies the following files for deployed Web Services:

- WSDL service descriptions
- Client-side proxy Jar (class files)
- Client-side proxy source

Oracle Application Server Web Services provides a Web Service Home Page for each deployed Web Service. To access a Home Page, enter a service endpoint of the form,

`http://host:port/context-root/service`

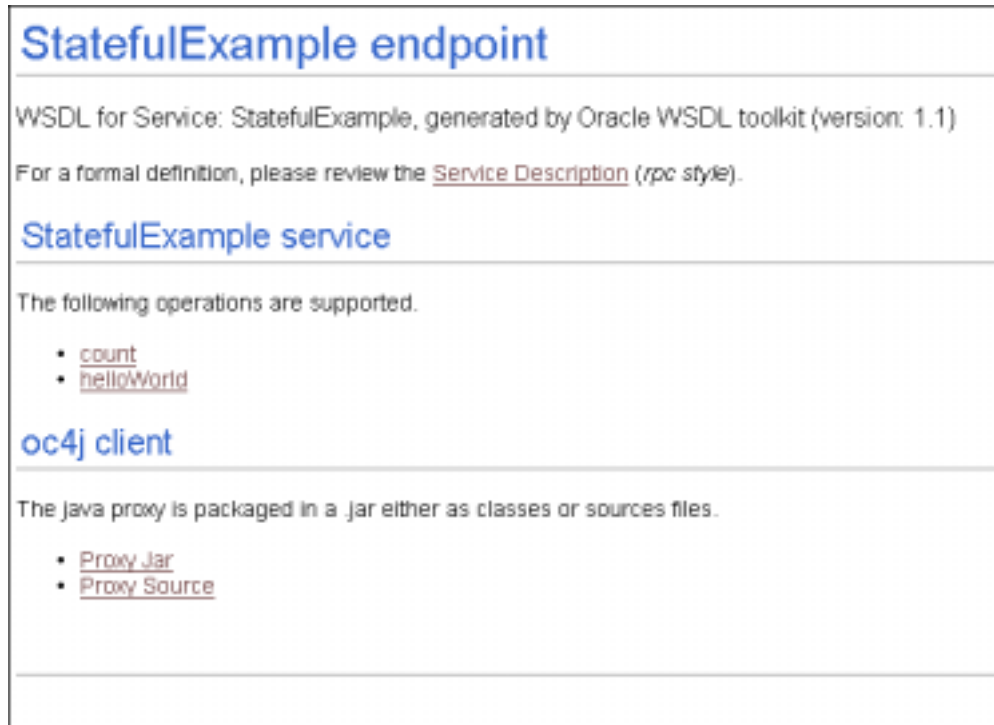
[Figure 8-1](#) shows the Web Service Home Page for StatefulExample, at the following endpoint,

`http://system1.us.oracle.com/webservices/statefulTest`

A Web Service Home Page provides the following:

- A Link to the WSDL file - To obtain the WSDL file for a Web Service, select the Service Description link and save the file.
- Links to Web Service Test Pages for each supported operation-To test the available Web Service operations enter the parameter values for the operation, if any, and select the Invoke button.
- Links to the Web Service client-side proxy Jar and the client-side proxy source - To obtain the client-side proxy Jar or the client-side proxy source, select the appropriate link, Proxy Jar or Proxy Source, and save the file.

Figure 8–1 Web Service Home Page



### Limitations for Web Service Test Pages

Web Service Test Pages have the following limitations:

- There is no support for complex input parameters for RPC style Web Services. Such pages do not support the Invoke button.
- There is no support for Document Style Web Services. Such pages do not support the Invoke button.

## Getting Web Service WSDL and Client-Side Proxies Directly

If you do not use the Web Service Home Page to get the WSDL file or client-side proxy for a Web Service, you can obtain these files directly.

This section covers the following:

- [Getting WSDL Service Descriptions](#)
- [Getting Client-Side Proxy Jar and Client-Side Proxy Source Jar](#)
- [Getting Client-Side Proxy Jar and Client-Side Proxy Source by Package](#)

### Getting WSDL Service Descriptions

To obtain the WSDL service description for a Web Service, use the Web Service URL and append a query string. The format for the URL to obtain the WSDL service description is as follows (see [Table 8–1](#) for a description of the URL components):

```
http://host:port/context-root/service?WSDL
```

or

```
http://host:port/context-root/service?wsdl
```

This command returns a WSDL description in the form `service.wsdl`. The `service.wsdl` description contains the WSDL for the Web Service named `service`, located at the specified URL. Using the WSDL that you obtain, you can build a client application to access the Web Service.

### Getting Client-Side Proxy Jar and Client-Side Proxy Source Jar

To obtain the client-side proxy Jar for a Web Service, use the Web Service URL and append a query string. The client-side proxy Jar file contains the proxy stubs class that supports building an application that communicates using SOAP to access the Web Service. The proxy class does the following:

- Provides a static location for the Web Service (the service does not need to be looked up in a UDDI registry).
- Provides proxy methods for each method exposed as part of the Web Service.
- Performs all of the work to construct the SOAP request, including marshalling and unmarshalling parameters, and handling the response.

The format for the URL to obtain the client-side proxy Jar is as follows (see [Table 8-1](#) for a description of the URL components):

```
http://host:port/context-root/service?PROXY_JAR
```

or

```
http://host:port/context-root/service?proxy_jar
```

This command returns the file `service_proxy.jar`. The `service_proxy.jar` is a Jar file that contains the client-side proxy classes that you can use to build a client-side application to access the Web Service.

To obtain the client-side proxy source Jar for a Web Service, use the Web Service URL and append a query string. The format for the URL to obtain the client-side proxy source Jar is as follows (see [Table 8-1](#) for a description of the URL components):

```
http://host:port/context-root/service?PROXY_SOURCE
```

or

```
http://host:port/context-root/service?proxy_source
```

This command returns the file `service_proxysrc.jar`. The file `service_proxysrc.jar` is a Jar file that contains the client-side proxy source files. This file represents the source code for the file `service_proxy.jar` associated with the service.

### Getting Client-Side Proxy Jar and Client-Side Proxy Source by Package

When you obtain the client-side proxy Jar file or the client-side proxy source Jar, you have the option of including a request parameter that specifies a package name for the generated client-side proxy classes or source files. If the Web Service's client-side Java class is part of a particular package, then you should specify the package name to match the client-side application's package name.

The format for the URL to obtain the client-side proxy Jar and specify the package name is as follows (see [Table 8-1](#) for a description of the URL components):

```
http://host:port/context-root/service?PROXY_JAR&packageName=mypackage
or
```

```
http://host:port/context-root/service?proxy_jar&packageName=mypackage
```

This command returns the file `service_proxy.jar`. The `service_proxy.jar` is a Jar file that contains the client-side proxy classes, using the specified package, `mypackage` for the Java package statement.

The format for the URL to obtain the client-side proxy source Jar and specify the package name is as follows (see [Table 8-1](#) for a description of the URL components):

```
http://host:port/context-root/service?PROXY_SOURCE&packageName=mypackage
or
```

```
http://host:port/context-root/service?proxy_source&packageName=mypackage
```

This command returns the file `service_proxysrc.jar`. As for the `proxy_jar`, you have the option of specifying a request parameter with a supplied package name by include a `packageName=name` option. The `service_proxysrc.jar` is a Jar file that contains the client-side source files for the client-side proxy that accesses the Web Service.

**Table 8-1 URL for Accessing Client Side Proxy Stubs**

URL Component	Description
<i>context-root</i>	The context-root is the value specified in the <code>&lt;context-root&gt;</code> tag for the web module associated with the Web Service. See the <code>META-INF/application.xml</code> in the Web Service's <code>.ear</code> file to determine this value.
<i>host</i>	This is the host of the Web Service's server running Oracle Application Server Web Services.
<i>mypackage</i>	This specifies the value that you want to use for the package name in the generated proxy Jar or proxy source.
<i>port</i>	This is the port of the Web Service's server running Oracle Application Server Web Services.
<i>service</i>	The service is the value specified in the <code>&lt;url-pattern&gt;</code> tag for the servlet associated with the Web Service. This is the service name. See the <code>WEB-INF/web.xml</code> in the Web Service's <code>.war</code> file to determine this value.

**See Also:**

- [Chapter 3, "Developing and Deploying Java Class Web Services"](#)
- [Chapter 4, "Developing and Deploying EJB Web Services"](#)
- [Chapter 5, "Developing and Deploying Stored Procedure Web Services"](#)

## Generating Client-Side Proxies With `WebServicesAssembler`

The Oracle Application Server Web Services `WebServicesAssembler` tool allows you to generate client-side proxies. A client-side proxy can access a Web Service that is deployed either on an Oracle Application Server Web Services endpoint or on a third party Web Service endpoint.

To generate a client-side proxy with `WebServicesAssembler`, specify a `<proxy-gen>` tag in the configuration file. [Table 8-2](#) describes the `<proxy-gen>` `WebServicesAssembler` configuration file sub-tags.

---

**Note:** When you are generating client-side proxies and you are accessing an external WSDL file from behind a firewall, make sure to set the appropriate security properties shown in [Table 8-4](#), such as `http.proxyHost` and `http.proxyPort`.

---

[Example 8-1](#) shows a sample `WebServicesAssembler` that includes a `<proxy-gen>` tag.

**Example 8-1** *WebServicesAssembler Proxy Gen Configuration File*

```
<?xml version="1.0"?>
<web-service>
  <proxy-gen>
    <proxy-dir>/TestArea/Hotel/proxy/outside</proxy-dir>
    <option name="include-source">true</option>
    <option name="wsdl-location" package-name="myPackage.proxy">
      http://terraservice.net/TerraService.asmx?WSDL</option>
    <option name="wsdl-location">
      http://ws.serviceobjects.net/sq/FastQuote.asmx?WSDL</option>
  </proxy-gen>
</web-service>
```

**Table 8-2** *Proxy Generation <proxy-gen> Sub-Tags*

Tag	Description
<code>&lt;proxy-dir&gt;</code> <i>directory</i> <code>&lt;/proxy-dir&gt;</code>	Specifies the directory for the generated client-side proxy stubs Jar file that is included in the generated Web Service .ear file.  This tag is required.
<code>&lt;option name="include-source"&gt;</code> <i>value</i> <code>&lt;/option&gt;</code>	Setting <i>value</i> to true tells <code>WebServicesAssembler</code> to include the classes and the source in the generated client-side proxy. When the value is false, the source is not included in the generated Jar.  This tag is optional.  Valid values: true, false  Default value: false
<code>&lt;option name="wsdl-location"&gt;</code> <i>URL</i> <code>&lt;/option&gt;</code> or <code>&lt;option name="wsdl-location"</code> <code>package-name="package"&gt;</code> <i>URL</i> <code>&lt;/option&gt;</code>	This tag sets the <i>URL</i> to use for the source WSDL to use to generate the client-side proxy.  This option also supports the optional attribute <code>package-name</code> . The <code>package-name</code> can specify the name <i>package</i> for the generated client-side proxy.  This tag is optional.  Examples:  <code>&lt;option name="wsdl-location"&gt;</code> <code>http://system1:8888/webservice3/TestService?WSDL</code> <code>&lt;/option&gt;</code>  <code>&lt;option name="wsdl-location"</code> <code>package-name="myPackage.proxy"&gt;</code> <code>http://system1:8888/webservice3/TestService?WSDL</code> <code>&lt;/option&gt;</code>

**See Also:** [Chapter 9, "Web Services Tools"](#)

## Working with Client-Side Proxy Jar to Use Web Services

This section describes how to use the client-side proxy Jar when you are building the client-side application to access a Web Service. The client-side proxy Jar class allows you to easily build an application that uses a Web Service.

The client side proxy Jar file contains a Java class to serve as a proxy to the Web Service implementation. The client-side proxy code constructs a SOAP request and marshalls and unmarshalls parameters for you. Using the proxy classes saves you the work of creating SOAP requests for accessing a Web Service or processing Web Service responses.

**Example 8-2** shows a source code sample client-side proxy extracted from a Web Service. For each operation available on the Web Service, there is a corresponding method in the proxy class. The example shows the method `helloWorld(String)` that serves as a proxy to the `helloWorld(String)` method in the associated Web Service implementation.

**Example 8-3** shows client-side application code that uses the `helloWorld()` method from the supplied client-side proxy shown in [Example 8-2](#).

---



---

**Note:** When you are accessing an external Web Service from behind a firewall, make sure to set the appropriate security properties shown in [Table 8-4](#), such as `http.proxyHost` and `http.proxyPort`.

---



---

### **Example 8-2** *Sample Client-side Proxy Method for Web Services*

```
public class StatefulExampleProxy {

    public java.lang.String helloWorld(java.lang.String param0) throws Exception
    {
        .
        .
        .
    }
    .
    .
    .
}
```

### **Example 8-3** *Sample Client-side Application Using a Proxy Class for Web Services*

```
import oracle.j2ee.ws_example.proxy.*;

public class Client
{
    public static void main(String[] argv) throws Exception
    {
        StatefulExampleProxy proxy = new StatefulExampleProxy();
        System.out.println(proxy.helloWorld("Scott"));
    }
}
```

```

    System.out.println(proxy.count());
    System.out.println(proxy.count());
    System.out.println(proxy.count());
}
}

```

## Setting the Web Services Proxy Client CLASSPATH

When you build a Web Services clients using a proxy, you need to use the correct CLASSPATH to run the client. [Table 8-3](#) lists jars that you need to include in the CLASSPATH.

**Table 8-3** Web Services CLASSPATH Components for a Client Using a Client-side Proxy

Component Jar	Description
<i>proxy.jar</i>	The <i>proxy</i> jar file that provides access to the Web Service.
<code>\$ORACLE_HOME/lib/xmlparserv2.jar</code>	The Oracle XML parser jar.
<code>\$ORACLE_HOME/j2ee/home/lib/http_client.jar</code>	The Oracle HTTP client jar.
<code>\$ORACLE_HOME/soap/lib/soap.jar</code>	The Oracle SOAP jar.
<code>\$ORACLE_HOME/j2ee/home/lib/mail.jar</code>	Generally, this is available in the JRE. If this is not available in the JRE, then include it in the CLASSPATH.
<code>\$ORACLE_HOME/j2ee/home/lib/activation.jar</code>	Generally, this is available in the JRE. If this is not available in the JRE, then include it in the CLASSPATH.
<code>\$ORACLE_HOME/jlib/javax-ssl-1_1.jar</code>	Used when the client uses SSL to connect to a Web Service that uses SSL. In this case, do not include <code>\$ORACLE_HOME/lib/jsee.jar</code> in the CLASSPATH.
<code>\$ORACLE_HOME/jlib/jsssl-1_1.jar</code>	Required when the client is using SSL to connect to a Web Service that uses SSL. In this case, either <code>\$ORACLE_HOME/jlib/javax-ssl-1_1.jar</code> or <code>\$ORACLE_HOME/lib/jsse.jar</code> must be specified.
<code>\$ORACLE_HOME/lib/jsse.jar</code>	Used when the client uses SSL to connect to a Web Service that uses SSL. In this case, do not include <code>\$ORACLE_HOME/jlib/javax-ssl-1_1.jar</code> in the CLASSPATH.
<code>\$ORACLE_HOME/webservices/lib/wsdl.jar</code>	Required when the client is using a Dynamic Proxy.
<code>\$ORACLE_HOME/webservices/lib/dsv2.jar</code>	Required when the client is using a Dynamic Proxy.

## Using Java Beans as Parameters for Web Services

When Java Beans are used as parameters to Oracle Application Server Web Services, the client-side code should use the generated Bean included with the downloaded client-side proxy. This is because the generated client-side proxy code translates SOAP structures to and from Java Beans by translating SOAP structure namespaces to and from fully qualified Bean class names. If a Bean with the specified name does not exist in the specified package, the generated client code will fail.

However, there is no special requirement for clients using Web Services Description Language (WSDL) to form calls to Oracle Application Server Web Services, rather than the client-side proxy. The generated WSDL document describes SOAP structures in a standard way. Application development environments, such as Oracle JDeveloper, which work directly from WSDL documents can correctly call Oracle Application Server Web Services with Java Beans as parameters.



## Using Web Services Security Features

When you run a client-side application that uses Oracle Application Server Web Services, you can access secure Web Services by setting properties in the client application. [Table 8-4](#) shows the available properties that provide credentials and other security information for Web Services clients. [Table 8-3](#) lists jar file that need to be included in the CLASSPATH, including those required to support SSL.

In a Web Services client application, you can set the security properties shown in [Table 8-4](#) as system properties by using the `-D` flag at the Java command line, or you can also set security properties in the Java program by adding these properties to the system properties (use `System.setProperty()` to add properties). In addition, the client side stubs include the `_setTransportProperties` method that is a public method in the client proxy stubs. This method enables you to set the appropriate values for security properties by supplying a `Properties` argument.

**Table 8-4 Web Services HTTP Transport Security Properties**

Property	Description
<code>http.authRealm</code>	Specifies the realm for which the HTTP authentication username/password is specified.  This property is mandatory when using basic authentication.
<code>http.authType</code>	Specifies the HTTP authentication type. The case of the value specified is ignored.  Valid values: <code>basic</code> , <code>digest</code>  The value <code>basic</code> specifies HTTP basic authentication.  Specifying any value other than <code>basic</code> or <code>digest</code> is the same as not setting the property.
<code>http.password</code>	Specifies the HTTP authentication password.
<code>http.proxyAuthRealm</code>	Specifies the realm for which the proxy authentication username/password is specified.
<code>http.proxyAuthType</code>	Specifies the proxy authentication type. The case of the value specified is ignored.  Valid values: <code>basic</code> , <code>digest</code>  Specifying any value other than <code>basic</code> or <code>digest</code> is the same as not setting the property.
<code>http.proxyHost</code>	Specifies the hostname or IP address of the proxy host.
<code>http.proxyPassword</code>	Specifies the HTTP proxy authentication password.
<code>http.proxyPort</code>	Specifies the proxy port. The specified value must be an integer. This property is only used when <code>http.proxyHost</code> is defined; otherwise this value is ignored.  Default value: 80
<code>http.proxyUsername</code>	Specifies the HTTP proxy authentication username.
<code>http.username</code>	Specifies the HTTP authentication username.

**Table 8–4 (Cont.) Web Services HTTP Transport Security Properties**

Property	Description
<code>java.protocol.handler.pkgs</code>	<p>Specifies a list of package prefixes for <code>java.net.URLStreamHandlerFactory</code>. The prefixes should be separated by " " vertical bar characters.</p> <p>This value should contain: <code>HTTPClient</code></p> <p>This value is required by the Java protocol handler framework; it is not defined by Oracle Application Server. This property must be set when using HTTPS. If this property is not set using HTTPS, a <code>java.net.MalformedURLException</code> is thrown.</p> <p><b>Note:</b> This property must be set as a system property.</p> <p>For example, set this property as shown in either of the following:</p> <ul style="list-style-type: none"> <li>■ <code>java.protocol.handler.pkgs=HTTPClient</code></li> <li>■ <code>java.protocol.handler.pkgs=sun.net.www.protocol   HTTPClient</code></li> </ul>
<code>oracle.soap.transport.allowUserInteraction</code>	<p>Specifies the allows user interaction parameter. The case of the value specified is ignored. When this property is set to <code>true</code> and either of the following are true, the user is prompted for a username and password:</p> <ol style="list-style-type: none"> <li>1. If any of properties <code>http.authType</code>, <code>http.username</code>, or <code>http.password</code> is not set, and a 401 HTTP status is returned by the HTTP server.</li> <li>2. If either of properties <code>http.proxyAuthType</code>, <code>http.proxyUsername</code>, or <code>http.proxyPassword</code> is not set and a 407 HTTP response is returned by the HTTP proxy.</li> </ol> <p>Valid values: <code>true</code>, <code>false</code></p> <p>Specifying any value other than <code>true</code> is considered as <code>false</code>.</p>
<code>oracle.ssl.ciphers</code>	<p>Specifies a list of: separated cipher suites that are enabled.</p> <p>Default value: The list of all cipher suites supported with Oracle SSL.</p>
<code>oracle.wallet.location</code>	<p>Specifies the location of an exported Oracle wallet or exported trustpoints.</p> <p><b>Note:</b> The value used is not a URL but a file location, for example:</p> <p><code>/etc/ORACLE/Wallets/system1/exported_wallet</code> (on UNIX)</p> <p><code>d:\oracle\system1\exported_wallet</code> (on Windows)</p> <p>This property must be set when HTTPS is used with SSL authentication, server or mutual, as the transport.</p>
<code>oracle.wallet.password</code>	<p>Specifies the password of an exported wallet. Setting this property is required when HTTPS is used with client, mutual authentication as the transport.</p>

## Working with WSDL Files and Oracle JDeveloper to Use Web Services

The Web Services WSDL allows you to manually, or using Oracle JDeveloper or another IDE, build client applications that use Web Services.

The Oracle JDeveloper IDE supports Oracle Application Server Web Services with WSDL features and provides unparalleled productivity for building end-to-end J2EE and integrated Web Services applications.

Oracle JDeveloper supports Oracle Application Server Web Services with the following features:

- Allows developers to create Java stubs from Web Services WSDL descriptions to programmatically use existing Web Services.

- Allows developers to create a new Web Service from Java or EJB classes, automatically producing the required deployment descriptor, web.xml, and WSDL file for you.
- Provides schema-driven WSDL file editing.
- Offers significant J2EE deployment support for Web Services J2EE .ear files, with automatic deployment to OC4J.

Non-Oracle Web Services IDEs or client development tools can use the supplied WSDL file to generate Web Services requests for services running under Oracle Application Server Web Services. Currently, many IDEs have the capability to create SOAP requests, given a WSDL description for the service.



---

---

## Web Services Tools

The Oracle Application Server Web Services assembly tool, `WebServicesAssembler`, assists in assembling Oracle Application Server Web Services. The Web Services assembly tool takes a configuration file which describes a Web Service, including the location of the Java classes, PL/SQL stored procedures or functions, or J2EE EAR, WAR, or JAR files and produces a J2EE EAR file that can be deployed under Oracle Application Server Web Services.

This chapter contains the following topics:

- [Running the Web Services Assembly Tool](#)
- [Web Services Assembly Tool Configuration File Sample](#)
- [Generating WSDL Files and Client Side Proxies](#)
- [Web Services Assembly Tool Configuration File Specification](#)
- [Web Services Assembly Tool Limitations](#)

### Running the Web Services Assembly Tool

Run the Web Services assembly tool as follows:

```
java -jar WebServicesAssembler.jar [-debug] -config [file]
or
java -jar WebServicesAssembler.jar [-debug]
```

Where *file* is a Web Services assembly tool configuration file. Without the `-config` option, a file named `config.xml` must be present in the same directory where `WebServicesAssembler.jar` is invoked.

With the `-debug` option, `WebServicesAssembler` displays verbose debugging comments.

---

---

**Note:** When running `WebServicesAssembler.jar` from the command line, the `PATH` environment variable should include the `JDK/bin` directory (the directory with the `javac` compiler).

---

---

### Web Services Assembly Tool Configuration File Sample

The sample configuration file shown in [Example 9-1](#) defines two services to be wrapped in an Enterprise ARchive file (EAR). The sample includes configuration information for services defined with `<stateless-java-service>` and `<stateful-java-service>` tags.

**See Also:**

- ["Preparing and Deploying Java Class Based Web Services"](#) on page 3-6
- ["Preparing and Deploying Stateless Session EJB Based Web Services"](#) on page 4-6
- ["Preparing Stored Procedure Web Services"](#) on page 5-2
- ["Preparing Document Style Web Services"](#) on page 6-6
- ["Preparing and Configuring JMS Web Services"](#) on page 7-8

**Example 9–1 Sample Web Services Assembly Tool Configuration File**

```
<web-service>

  <display-name>Web Services Example</display-name>
  <description>Java Web Service Example</description>
  <!-- Specifies the resulting web service archive will be stored in ./ws_example.ear -->
  <destination-path>./ws_example.ear</destination-path>
  <!-- Specifies the temporary directory that web service assembly
       tool can create temporary files. -->
  <temporary-directory>./tmp</temporary-directory>
  <!-- Specifies the web service will be accessed in the servlet context
       named "/webservices". -->
  <context>/webservices</context>

  <!-- Specifies the web service will be stateless -->
  <stateless-java-service>
    <interface-name>oracle.j2ee.ws_example.StatelessExample</interface-name>
    <class-name>oracle.j2ee.ws_example.StatelessExampleImpl</class-name>
    <!-- Specifies the web service will be accessed in the uri named
         "statelessTest" within the servlet context. -->
    <uri>/statelessTest</uri>
    <!-- Specifies the location of Java class files are under ./src -->
    <java-resource>./src</java-resource>
  </stateless-java-service>

  <stateful-java-service>
    <interface-name>oracle.j2ee.ws_example.StatefulExample</interface-name>
    <class-name>oracle.j2ee.ws_example.StatefulExampleImpl</class-name>
    <!-- Specifies the web service will be accessed in the uri named
         "statefulTest" within the servlet context. -->
    <uri>/statefulTest</uri>
    <!-- Specifies the location of Java class files are under ./src -->
    <java-resource>./src</java-resource>
  </stateful-java-service>

</web-service>
```

**Web Services Assembly Tool Configuration File Sample Output**

After running the Web Services Assembly tool with the sample input file shown in [Example 9–1](#), the generated output is an EAR file (`/tmp/ws_example.ear`). The generated J2EE .ear file, `ws_example.ear`, has the structure shown in [Example 9–2](#).

**Example 9–2 Structure of Web Services Assembly Tool Sample Ear File**

```

ws_example.ear
|---META-INF
|   `---application.xml
`---ws_example_web.war
    |---index.html
    `---WEB-INF
        |-----web.xml
        `-----classes
            `-----oracle
                `-----j2ee
                    `---ws_example
                        |---StatefulExample.java
                        |---StatefulExample.class
                        |---StatefulExampleImpl.java
                        `---StatefulExampleImpl.class
                        |---StatelessExample.java
                        |---StatelessExample.class
                        |---StatelessExampleImpl.java
                        `---StatelessExampleImpl.class

```

## Generating WSDL Files and Client Side Proxies

This section describes using the `<wsdl-gen>` and `<proxy-gen>` tags in a `WebServicesAssembler` configuration file. These tags controls the options for generating WSDL files and client-side proxies for Web Services. A client-side developer can obtain and use the WSDL file or the client-side proxies to build an application that uses a Web Service. A server-side developer that is assembling Web Services can use these file for testing Web Services.

This section covers the following topics:

- [Generating and Assembling WSDL Files](#)
- [Generating Client-Side Proxies with WSDL](#)

## Generating and Assembling WSDL Files

Using Oracle Application Server Web Services, a Web Service developer has several choices for deciding how the WSDL file that is associated with a Web Service is generated:

1. Using the `<wsdl-gen>` tag, you can specify that `WebServicesAssembler` create the WSDL file. At assembly time, when the Web Service is prepared, the `WebServicesAssembler` generates and packages the WSDL file with the Web Service.

[Example 9–3](#) shows a configuration file that includes the `<wsdl-gen>` tag.

2. Allowing the Oracle Application Server Web Services runtime to generate the WSDL file when the WSDL is requested by a Web Service client (after the WEB Service is deployed). In this case, you do not specify the `<wsdl-gen>` tag in the configuration file.
3. Creating a WSDL file manually. In this case, use the `<wsdl-gen>` tag during assembly of the J2EE .ear file to specify the path to the WSDL file. At assembly

time when the Web Service is prepared, the `WebServicesAssembler` packages the WSDL file with the Web Service.

**Table 9-1** describes the `<wsdl-gen>` `WebServicesAssembler` configuration file sub-tags.

---

**Note:** Using the `<wsdl-gen>` tag, the default behavior is to package the WSDL into the J2EE .ear file. To exclude the generated WSDL from the J2EE .ear file, use `<option name="packageIt">` tag and set the value to `false`.

---

**Table 9-1 WSDL Generation <wsdl-gen> Sub-Tags**

Tag	Description
<code>&lt;option name="force"&gt;</code> <i>value</i> <code>&lt;/option&gt;</code>	Setting <i>value</i> to <code>true</code> forces <code>WebServicesAssembler</code> to overwrite any existing WSDL file in the WSDL directory specified with the <code>&lt;wsdl-dir&gt;</code> tag.  Valid values: <code>true</code> , <code>false</code>  Default value: <code>true</code>
<code>&lt;option name="httpServerURL"&gt;</code> <i>URL</i> <code>&lt;/option&gt;</code>	This tag sets the value for the HTTP server listener endpoint in the generated WSDL. Set the <i>URL</i> to point to the Web Service HTTP listener.  Example: <code>&lt;option name="httpServerURL"&gt;http://localhost:8888&lt;/option&gt;</code>
<code>&lt;option name="packageIt"&gt;</code> <i>value</i> <code>&lt;/option&gt;</code>	Setting <i>value</i> to <code>true</code> tells <code>WebServicesAssembler</code> to include the generated WSDL in the assembled .ear file. When the value is <code>false</code> , the generated WSDL file is not included in the assembled .ear file.  Valid values: <code>true</code> , <code>false</code>  Default value: <code>true</code>
<code>&lt;wsdl-dir&gt;</code> <i>directory</i> <code>&lt;/wsdl-dir&gt;</code>	Specifies the <i>directory</i> for the WSDL file source that is included in the generated Web Service .ear file.  When you are manually supplying the WSDL file, place a copy of the WSDL file in the specified directory and use the <code>&lt;option name="force"&gt;</code> tag with the value <code>false</code> .

**Example 9-3 WebServicesAssembler Configuration File Including <wsdl-gen>**

```
<web-service>

  <display-name>Stateless Java Document Web Service</display-name>
  <description>Stateless Java Document Web Service Example</description>
  <destination-path>./statelessdocws.ear</destination-path>
  <temporary-directory>./temp</temporary-directory>
  <context>/statelessdocws</context>
  <option name="source-path">converter.xsl</option>

  <stateless-java-service>
    <interface-name>StatelessDoc</interface-name>
    <class-name>StatelessDocImpl</class-name>
    <uri>/docservice</uri>
    <java-resource>./classes</java-resource>
    <message-style>doc</message-style>
```



```

</stateless-java-service>

<!-- generate the wsdl -->
<wsdl-gen>
  <wsdl-dir>wsdl</wsdl-dir>
  <!-- over-write a pregenerated wsdl , turn it 'false'
       to use the pregenerated wsdl-->
  <option name="force">true</option>
  <option name="httpServerURL">http://localhost:8888</option>
</wsdl-gen>

</web-service>

```

### Manually Producing a WSDL File

When you do not want to use either the `WebServicesAssembler` tool generated WSDL or the Oracle Application Server Web Services runtime generated WSDL file, and you want to supply your own version of the Web Service WSDL file, perform the following steps:

1. Manually create the WSDL file for your service.
2. Name the WSDL file with a name using the `.wsdl` extension placed after the service name. For example, `service1.wsdl` for a service named `service1`.
3. Create a configuration file that includes the `<wsdl-gen>` tag, including `<option name="force">` set to `false` and `<option name="packageIt">` set to `true`.
4. Place the WSDL file that you create in the directory specified with the `<wsdl-dir>` tag.
5. Run the `WebServicesAssembler` with the specified configuration file.

## Generating Client-Side Proxies with WSDL

When the `<proxy-gen>` tag is included in a configuration file with the `<wsdl-gen>`, the generated WSDL is used to generate the proxy that is placed in the specified directory (this occurs when `WebServicesAssembler` runs during the Web Service assembly process).

[Table 8-2](#) lists the `<proxy-gen>` sub-tags.

---



---

**Note:** Using `<proxy-gen>`, the generated proxy is not assembled in the J2EE `.ear` file.

---



---

[Example 9-4](#) shows a sample configuration file that includes both the `<wsdl-gen>` and the `<proxy-gen>` tags.

#### **Example 9-4** *WebServicesAssembler Configuration File Including `<wsdl-gen>`*

```

<web-service>
  <display-name>Test</display-name>
  <description>Test program</description>
  <destination-path>test.ear</destination-path>
  <temporary-directory>temp/</temporary-directory><context>/HotelService</context>
  <option name="source-path">Workspace1/common/classes</option>

  <stateless-java-service>

```

```
<interface-name>com.mypackage1.Itest</interface-name>
<uri>/main</uri>
<class-name>com.mypackage1.test</class-name>
</stateless-java-service>

<wsdl-gen>
  <wsdl-dir>wsdl</wsdl-dir>
  <option name="force">true</option>
  <option name="httpServerURL">http://localhost:8888</option>
  <option name="packageIt">false</option>
</wsdl-gen>

<proxy-gen>
  <proxy-dir>proxy</proxy-dir>
  <option name="include-source">true</option>
</proxy-gen>

</web-service>
```

## Web Services Assembly Tool Configuration File Specification

The input file for `WebServicesAssembler` is an XML file conforming to the Web Services Assembly Tool configuration file DTD.

[Example 9-5](#) shows the Web Services Assembly Tool Configuration file DTD.

### **Example 9-5 Assembly Tool Input File DTD**

```
<?xml version="1.0" encoding="UCS-2"?>
<!-- Specify the properties of the web services to be assembled. -->
<!ELEMENT web-service
((display-name)?,(description)?,destination-path,temporary-directory,context,(datasource-JNDI-name)?,(stateful-
java-service)*,(stateless-java-service)*,(stateless-stored-procedure-java-service)*,(stateless-session-ejb-servic
e)*,(jms-doc-service)*,(option)*,(wsdl-gen)?,(proxy-gen)?)>
<!ELEMENT display-name (#PCDATA)*>
<!ELEMENT description (#PCDATA)*>
<!-- Specify the full path of the resulting EAR file. For example,
"/home/demo/webservices.ear" -->
<!ELEMENT destination-path (#PCDATA)*>
<!-- Specify a directory where the assembly tool can create temporary
directories and files. -->
<!ELEMENT temporary-directory (#PCDATA)*>
<!-- Specify the context root of the web services. For example, "/webservices". -->
<!ELEMENT context (#PCDATA)*>
<!-- for specifying database resource refs -->
<!ELEMENT datasource-JNDI-name (#PCDATA)*>

<!-- Specify the properties of a stateful Java service -->
<!ELEMENT stateful-java-service
((interface-name)?,class-name,uri,(java-resource)*,(ejb-resource)*,(scope)*,(session-timeout)*,(message-style)?
>
<!-- Specify the properties of a stateless Java service -->
<!ELEMENT stateless-java-service
((interface-name)?,class-name,uri,(java-resource)*,(ejb-resource)*,(message-style)?)>
<!-- Specify the properties of a stateless stored procedure Java service -->
<!ELEMENT stateless-stored-procedure-java-service
((interface-name)?,(class-name)?,uri,database-JNDI-name,(java-resource)?,(jar-generation)?)>
<!-- Specify the properties of a stateless session ejb service -->
<!ELEMENT stateless-session-ejb-service (path,uri,ejb-name,(ejb-resource)*)>

<!-- Specify the java interface which defines the public methods to be exposed
in the web service. For example, "com.foo.myproject.helloWorld". -->
<!ELEMENT interface-name (#PCDATA)*>
```

```

<!-- Specify the java class to be exposed as a web service. If interface-name is
not specified, all the public methods in this class will be exposed. For example,
    "com.foo.myproject.helloWorldImpl". -->
<!ELEMENT class-name (#PCDATA)*>
<!-- Specify the uri of this service. This uri is used in the URL to access the
WSDL and client jar, and invoke the web service. For example, "/myService". -->
<!ELEMENT uri (#PCDATA)*>
<!--
Specify the java resources used in this service.
The value can be a directory or a file that implements the web services. If it
is a directory, all the files and subdirectories under the directory are copied
and packaged in the Enterprise ARchive. If the java resource should belong to a
java package, you should either package it as a jar file and specify it as a
java resource, or create the necessary directory and specify the directory which
contains this directory structure as java resource. For example, you want to
include "com.mycompany.mypackage.foo" class as a java resource of the web
services, you can either package this class file in foo.jar and specify
<java-resource>c:/mydir/foo.jar</java-resource>, or place the class under
d:/mydir/com/mycompany/mypackage/foo.class and specify the java resource as
<java-resource>c:/mydir/</java-resource>.
-->
<!ELEMENT java-resource (#PCDATA)*>
<!-- Specify the ejb resources used in this service. ejb-resource should be a
jar file that implements a enterprise java bean. -->
<!ELEMENT ejb-resource (#PCDATA)*>
<!-- Specify the database JNDI name for stateless PL/SQL web service.-->
<!ELEMENT database-jndi-name (#PCDATA)*>
<!-- Specifies the path of the EJB jar file to exposed as web services. -->
<!ELEMENT path (#PCDATA)*>
<!-- Specify the ejb-name of the session bean to be exposed as web services.
ejb-name should match the <ejb-name> value in the META-INF/ejb-jar.xml of the bean. -->
<!ELEMENT ejb-name (#PCDATA)*>
<!-- Specify scope of Stateful Java service -->
<!ELEMENT scope (#PCDATA)*>
<!-- Specify session timeout of Stateful Java service -->
<!ELEMENT session-timeout (#PCDATA)*>
<!-- Specify the directory location of the generated wsdl-->
<!ELEMENT wsdl-dir (#PCDATA)*>
<!-- Specify that wsdl generation is to happen 'force' 'httpServerURL' 'packageIt'-->
<!ELEMENT wsdl-gen (wsdl-dir,(option)*)>
<!-- Specifyg the directory location of the generated proxy-->
<!ELEMENT proxy-dir (#PCDATA)*>
<!ELEMENT option (#PCDATA)*>
<!ATTLIST option name CDATA #REQUIRED>

<!-- Specifying that proxy generation is asked for , it can have optional tags as
'include-source' 'wsdl-location' -->
<!ELEMENT proxy-gen (proxy-dir,(option)*)>
<!ELEMENT jar-generation (db-package-name,db-schema,db-url,prefix,(method-name)*)>
<!ELEMENT database-JNDI-name (#PCDATA)*>
<!ELEMENT db-package-name (#PCDATA)*>
<!ELEMENT db-url (#PCDATA)*>
<!ELEMENT db-schema (#PCDATA)*>
<!ELEMENT prefix (#PCDATA)*>
<!ELEMENT method-name (#PCDATA)*>
    <!-- specify the message style ,if this tag is not present it is considered to have 'rpc' ..it can have values
of 'rpc' or 'doc' or 'document' -->
<!ELEMENT message-style (#PCDATA)*>

<!ELEMENT connection-factory-resource-ref (#PCDATA)*>
<!ELEMENT topic-resource-ref (#PCDATA)*>
<!ELEMENT queue-resource-ref (#PCDATA)*>
<!--Resource ref of the return destination factory-->
<!ELEMENT reply-to-connection-factory-resource-ref (#PCDATA)*>
<!--Resource ref of the return destination Topic. -->
<!ELEMENT reply-to-topic-resource-ref (#PCDATA)*>

```

```
<!--Resource ref of the return destination Queue. -->
<!ELEMENT reply-to-queue-resource-ref (#PCDATA)*>
<!--jms-priority ,jms-message-type,jms-delvery-mode ,jms-expiration The JMS properties are only set for
enqueueing operations, i..e, for send operations only. -->
<!ELEMENT jms-priority (#PCDATA)*>
<!ELEMENT jms-message-type (#PCDATA)*>
<!ELEMENT jms-delivery-mode (#PCDATA)*>
<!ELEMENT jms-expiration (#PCDATA)*>
<!-- operation property is optional. Possible values for this parameter are: send, receive, and both. If not
provided, the value defaults to both. -->
<!ELEMENT operation (#PCDATA)*>
<!ELEMENT jms-doc-service
(uri,connection-factory-resource-ref,(topic-resource-ref)?,(queue-resource-ref)?,(reply-to-connection-factory-re
source-ref)?,(reply-to-topic-resource-ref)?,(reply-to-queue-resource-ref)?,(jms-priority)?,(jms-message-type)?,(
jms-delivery-mode)?,(jms-expiration)?,(operation)?)>
```

## Web Services Assembly Tool Limitations

The `WebServicesAssembler` tool has the following limitations:

- **No Upload/download capabilities:** the Web Services Assembly tool does not upload Java classes from a client system to a server or download a generated EAR file back to a client system.
- **Does not support advanced configuration tasks:** for example, the Web Services Assembly tool is not able to control the security options for a Web Services Servlet, cannot secure an EJB, secure welcome files, or perform other administrative tasks.

---

## Discovering and Publishing Web Services

Oracle Application Server Containers for J2EE (OC4J) provides a Universal Discovery Description and Integration (UDDI) Web Services registry known as Oracle Application Server UDDI Registry (OracleAS UDDI Registry).

With OracleAS UDDI Registry, Web Services provider administrators in an enterprise environment can publish their Web Services for use by Web Services consumers (application programmers). Web Services consumers can use the UDDI inquiry interface to discover published Web Services and can use those services in their applications for a particular enterprise process.

This chapter is organized into the following main sections:

- [Understanding a UDDI Registry](#)
- [Introducing OracleAS UDDI Registry](#)
- [Getting Started with OracleAS UDDI Registry](#)
- [Web Services Discovery](#)
- [Web Services Publishing](#)
- [OracleAS UDDI Registry Administration](#)
- [UDDI Open Database Support](#)
- [OracleAS UDDI Registry Server Error Messages](#)
- [Command-Line Options for the uddiadmin.jar Tool](#)
- [Server Configuration Properties](#)

### Understanding a UDDI Registry

The information provided in a UDDI registry can be used to perform three types of searches:

- **White pages search**—address, contact, and known identifiers. For example, search for a business that you already know something about, such as its name or some unique identifier (ID).
- **Yellow pages topic search**—industrial categories based on standard classifications, such as NAICS, ISO-3166, and UNSPSC.
- **Green pages service search**—technical information about Web Services that are exposed by a business, including references to specifications of interfaces for Web Services, as well as support for pointers to various file and URL-based discovery mechanisms.

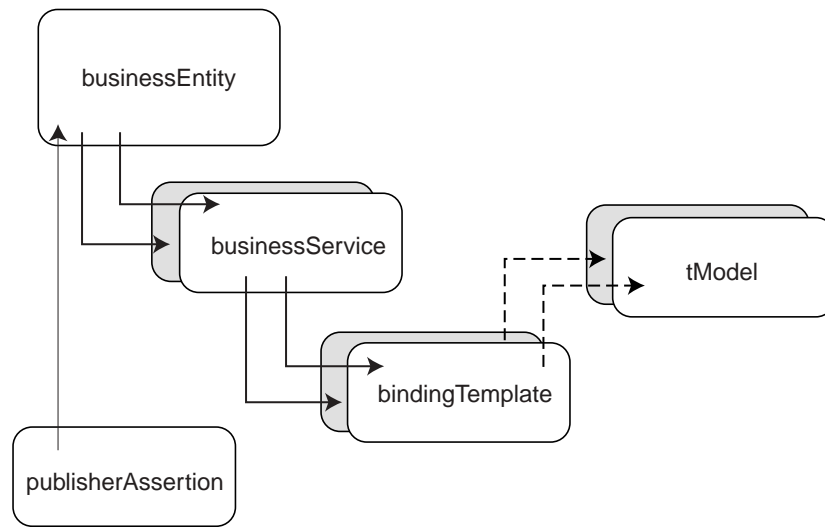
A UDDI registry uses standards-based technologies, such as common Internet protocols (TCP/IP and HTTP), XML, and SOAP, which is a specification for using XML in simple message-based exchanges. UDDI is a standard Web Services description format and Web Services discovery protocol; a UDDI registry can contain metadata for any type of service, with best practices already defined for those described by Web Services Description Language (WSDL).

## UDDI Registry Data Structure

The UDDI registry consists of the following five data structure types, which group information to facilitate rapid location and comprehension of registry information:

- **businessEntity**—the top-level, logical parent data structure. It contains descriptive information about the business that publishes information about Web Services, such as company name, contacts, business services, categories, contacts, discovery URLs, and identifier and category information that is useful for performing searches.
- **businessService**—the logical child of a single **businessEntity** data structure as well as the logical parent of a **bindingTemplate** structure. It contains descriptive business service information about a particular family of technical services including its name, brief description, technical service description, and category information that is useful for performing searches.
- **bindingTemplate**—the logical child of a single **businessService** data structure. It contains technical information about a Web Services entry point, and references to interface specifications.
- **tModel**—a description of specifications for Web Services, or a classification that forms the basis for technical identification. It represents the technical specification of Web Services. It facilitates the Web Services consumers (programmers) searching for registered Web Services that are compatible with a particular technical specification. That is, based on the descriptions of the specifications for Web Services in the **tModel** data structure, Web Services consumers can easily identify other compatible Web Services.
- **publisherAssertion**—information about a relationship between two parties, asserted by one or both.

[Figure 10-1](#) shows the UDDI information model and the relationships among its five data structure types.

**Figure 10-1 UDDI Registry Information Model**

Because the UDDI registry makes use of XML and SOAP, each of these data structures contains a number of elements and attributes that further serve to describe a business or have a technical purpose.

For a complete description of the UDDI service description framework, see *UDDI Version 2.03, Data Structure Reference Published Specification, Dated 19 July 2002* and *UDDI Version 2.04 API, Published Specification Dated 19 July 2002*, available at:

<http://www.uddi.org/specification.html>

## Introducing OracleAS UDDI Registry

Using OracleAS UDDI Registry in an enterprise environment, Web Services provider administrators can publish their Web Services for use by Web Services consumers (programmers). Web Services consumers can use the UDDI inquiry interface to discover published Web Services by browsing, searching, and drilling down in OracleAS UDDI Registry. Consumers can select one or more Web Services from among those registered, and use those services in their applications for a particular enterprise process.

For example, an administrator can publish the Web Services by providing all the metadata and pointers to the interface specification in OracleAS UDDI Registry. The administrator can work with consumers who have completed a Web Services implementation using the J2EE stack (Enterprise JavaBeans (EJB), JavaBeans, JavaServer Pages (JSP), or servlets) and expose the implementation as Web Services based on Simple Object Access Protocol (SOAP). In this way, the administrator publishes the availability of these Web Services for the Web Services consumers to discover and select for use in their own applications.

OracleAS UDDI Registry is compliant with the UDDI Version 2 specifications, including the following specifications:

- UDDI Version 2.04 API Specification
- UDDI Version 2.03, Data Structure Reference Specification
- UDDI Version 2.03, Replication Specification

OracleAS UDDI Registry support for Web Services deployed in OC4J is composed of the following parts:

- Web Services discovery—Consumers can use the Inquiry API to implement their own Web Services discovery tool to search, locate, and drill down to discover J2EE Web Services in OracleAS UDDI Registry, as well as in any other accessible UDDI registry compatible with the UDDI v1.0 or V2.0 specifications. See "[Using the OracleAS UDDI Registry Inquiry API](#)" on page 10-10 for more information about using the Inquiry API and locating the Javadoc documentation.
- Web Services publishing—Administrators can deploy J2EE Web Services using Oracle Enterprise Manager 10g. As part of the deployment process, the administrator can publish Web Services to OracleAS UDDI Registry.  
  
Consumers can use the Publishing API to publish Web Services by providing save and delete calls for each of the five key UDDI data structures (businessEntity, businessService, bindingTemplate, tModel, and publisherAssertion). See "[Using the OracleAS UDDI Registry Publishing API](#)" on page 10-28 for more information about using the Publishing API and locating the Javadoc documentation.
- Web Services updates—Administrators can update published Web Services by searching, locating, and drilling down to J2EE Web Services using Oracle Enterprise Manager 10g.
- Additional tool development—Consumers can use a Java-based client library to facilitate additional tool development and application development. See the *Oracle Application Server Web Services UDDI Client API Reference* for information about this API.
- Replication management—Administrators can create a logical registry that comprises one or more OracleAS UDDI Registry implementations and UDDI implementations from other vendors that also implement the UDDI v2.03 Replication Specification. See "[UDDI Replication](#)" on page 10-41 for more information.
- Database support—UDDI open database support is provided for Microsoft SQL Server, IBM DB2, and Oracle (non-infrastructure) databases. See "[UDDI Open Database Support](#)" on page 10-50 for more information.

## Support for Standard Classification and Identifier Systems

OracleAS UDDI Registry supports standard taxonomies for classifying (categorizing) tModels, businessEntities, and businessServices and for identifying tModels and BusinessEntities.

OracleAS UDDI Registry provides the following built-in and checked standard taxonomies for categorizing tModels, businessEntities, and businessServices:

- North American Industry Classification System (NAICS) 1997 Release  
  
This is a classification system for each industry and corresponding code. For more information about NAICS, see the following Web site:  
  
<http://www.census.gov/epcd/www/naics.html>
- United Nations Standard Products and Services Codes (UNSPSC) Version 7.3  
  
This is the first coding system to classify both products and services for use throughout the global marketplace. For more information about UNSPSC, following Web site:  
  
<http://www.unspsc.org/>
- ISO-3166 Geographic Code System (ISO-3166)



This a list of all countries and their subdivisions. For more information about ISO-3166, see the following Web site:

<http://www.iso.org/iso/en/prods-services/iso3166ma/02iso-3166-code-lists/index.html>

When administrators publish Web Services, they can select the taxonomy and the category to which they want to register the Web Services. They have the option of publishing their Web Services to any or all three of these taxonomies, and to as many categories and subcategories as they wish within each.

OracleAS UDDI Registry also provides the following built-in identifier systems for identifying tModels and businessEntities:

- Dun & Bradstreet D-U-N-S Number Identifier System (D-U-N-S)

This system uses unique nine-digit sequences for identifying businesses worldwide. For more information, see the following Web site:

<http://www.dnb.com>

- Thomas Register Supplier Identifier Code System

This system provides identification codes for manufacturers and suppliers. For more information see the following Web site:

<http://www.thomasregister.com/>

**Table 10-1** lists the taxonomy, its name, and its tModel key, a unique universal identifier (UUID).

**Table 10-1** *Classifications and Identifier Taxonomies*

Taxonomy	Name	tModel key
NAICS	ntis-gov:naics:1997	uuid:C0B9FE13-179F-413D-8A5B-5004DB8E5BB2
UNSPSC	unspsc-org:unspsc	uuid:CD153257-086A-4237-B336-6BDCBDCC6634
ISO-3166	uddi-org:iso-ch:3166-1999	uuid:4E49A8D6-D5A2-4FC2-93A0-0411D8D19E88
D-U-N-S	dnb-com:D-U-N-S	uuid:8609C81E-EE1F-4D5A-B202-3EB13AD01823
Thomas Register	thomasregister-com:supplierID	uuid:B1B1BAF5-2329-43E6-AE13-BA8E97195039

For more information about the taxonomies, see the following Web site:

[http://www.uddi.org/taxonomies/UDDI\\_Taxonomy\\_tModels.htm](http://www.uddi.org/taxonomies/UDDI_Taxonomy_tModels.htm)

## UUID Generation

OracleAS UDDI Registry uses an algorithm to generate a version 4 Unique Universal Identifier (UUID) from random numbers.

All built-in tModel data structures as specified in the UDDI v2 specification are included. An additional tModel data structure `uddi-org:operators`, defined in the UDDI v2 specification, is also included to classify the bootstrap node `businessEntity` that represents OracleAS UDDI Registry itself.

## Getting Started with OracleAS UDDI Registry

This section describes how to get started using OracleAS UDDI Registry. It includes the following topics:

- [Configuring OracleAS UDDI Registry](#)
- [Modifying Properties at Installation or First-Use](#)
- [Considerations in a Production Environment](#)

## Configuring OracleAS UDDI Registry

If you install the Oracle Application Server Infrastructure and the OracleAS Portal and Wireless middle tier, OracleAS UDDI Registry is installed. OracleAS UDDI Registry is automatically deployed into an OC4J\_Portal instance and the UDDI database schema is embedded in the OracleAS Infrastructure database.

If you want to install OracleAS UDDI Registry with OC4J standalone or with Oracle Application Server Core install, please refer to the standalone OracleAS UDDI Registry kit on OTN:

<http://www.oracle.com/technology/tech/webservices/htdocs/uddi>

To initialize and configure OracleAS UDDI Registry, you must access (either through the browser or programmatically through a SOAP invocation) the UDDI servlet inquiry end point or publishing SOAP end points. Otherwise, you will not be able to use OracleAS UDDI Registry from Oracle Enterprise Manager 10g, including the integrated Web Services publishing.

To initialize and configure OracleAS UDDI Registry by pinging the UDDI inquiry servlet end point from a browser, enter the following URL:

`http://OracleAS-host:OracleAS-port/uddi/inquiry`

The OracleAS UDDI Registry page is displayed. You should see the message: "Welcome! Your registry is now up and running."

This initialization and configuration step sets up the following:

- UDDI core tModel data structures
- A businessEntity node representing the registry node
- The businessEntity discoveryURL prefix and the operatorName property

By default, the installation creates UDDI users and user groups. [Table 10-2](#) lists the type of user, the user names, and passwords.

**Table 10-2** *Default UDDI Users and Passwords*

Type	User Name	Default Password
Administration	ias_admin	ias_admin123
Publisher	uddi_publisher	uddi_publisher123
Publisher	uddi_publisher1	uddi_publisher1
Replicator	uddi_replicator	no password, not used explicitly

You can connect to the UDDI end points using these user names and passwords. "[Managing Users](#)" on page 10-36 provides more information about the UDDI users and groups that are set up during installation.

The OracleAS UDDI Registry is available through the following URLs:

- Getting started information

`http://OracleAS-host:OracleAS-port/uddi/`

- UDDI inquiry SOAP end point  
`http://OracleAS-host:OracleAS-port/uddi/inquiry`
- UDDI publishing SOAP end point  
`http://OracleAS-host:OracleAS-port/uddi/publishing`
- UDDI administration end point  
`http://OracleAS-host:OracleAS-port/uddi/admin`
- UDDI replication SOAP end point  
`http://OracleAS-host:OracleAS-port/uddirepl/replication`
- UDDI replication HTTPS Wallet Password Administration end point  
`http://OracleAS-host:OracleAS-port/uddirepl/admin/wallet`

## Modifying Properties at Installation or First-Use

You perform many administrative operations using the command-line tool `uddiadmin.jar`. The tool is located in the `uddi/lib/uddiadmin.jar` file for UNIX and in the `uddi\lib\uddiadmin.jar` file for Windows. In general, the command-line tool uses the following format:

```
java -jar uddiadmin.jar http://OracleAS-host:port/uddi/admin username password
[-verbose] options_and_their_parameters
```

With the `setProperty` option of the command-line tool `uddiadmin.jar`, you can set the value of configuration properties, such as maximum database connections and default language.

You should change the following two properties immediately after installation. However, once you change them, you should not change them again unless there are changes to the host setup or you move the system from a staging environment to a production environment.

- `operatorName`: Provides the name of the operator of OracleAS UDDI Registry. This name appears in the operator attribute of responses. Setting this property applies in a retroactive fashion to existing entities in the database. For example, changing the operator name results in the new operator name replacing the old operator name in all business and tModel data structures.

The following example sets the `operatorName` property to `OracleUddiServerIT_Dept`:

```
java -jar uddiadmin.jar http://OracleAS-host:port/uddi/admin username password
-setProperty oracle.uddi.server.operatorName=OracleUddiServerIT_Dept
```

- `businessEntityURLPrefix`: Provides the prefix of the generated discoveryURL, which is automatically generated for each `businessEntity` data structure saved in the registry. The prefix should be customized for your deployment environment. Setting this parameter applies in a retroactive fashion to existing entities in the database. For example, changing the discoveryURL prefix results in the new discoveryURL replacing the old discoveryURLs in all `businessEntity` usetypes.

The host name and port should be the host name and port of the Web server (which may or may not be the same as the servlet container).

The following example sets the prefix of the discoveryURL to `http://uddihost:port/uddi/inquiryget`:

```
java -jar uddiadmin.jar http://OracleAS-host:port/uddi/admin username password
-setProperty
oracle.uddi.server.businessEntityURLPrefix=http://uddihost:port/uddi/inquiryget
```

---

**Note:** Be sure to set these properties before enabling UDDI replication.

---

In addition, you can set the default language of the registry by using the `defaultLang` property. See ["defaultLang"](#) on page 10-71 for more information.

See ["Using the Command-Line Tool uddiadmin.jar"](#) on page 10-34 for information about the command-line tool. See ["setProperty"](#) on page 10-66 for more information about the `setProperty` option.

## Considerations in a Production Environment

The following information describes some postinstallation configuration steps that you should do immediately after the installation. These steps are not mandatory, but are highly recommended in a production environment:

- **Security for publishing the end point:** By default, HTTP access is enabled. However, HTTPS access is recommended for security concerns. See ["Transport Security"](#) on page 10-50 for more information about disabling HTTP access.
- **Database connection pool sizing and statement caching:** Database connection pool parameters, such as maximum number of database connections and usage of statement caching, should be configured to accommodate the actual database server load.

If you are using an Oracle database other than the OracleAS Infrastructure database as the back-end storage, you can configure the parameters by editing the data source `jdbc/OracleUddi`. Refer to the chapter on data sources in the *Oracle Application Server Containers for J2EE Services Guide* for more information.

If you are using the OracleAS Infrastructure database as the back-end storage, you can configure the parameters by modifying the following UDDI server configuration properties:

- [minConnections](#)
- [maxConnections](#)
- [jdbcDriverType](#)
- [stmtCacheType](#)
- [stmtCacheSize](#)
- **Change of the operatorName and businessEntity discoveryURL prefix:** In some cases, you may want to change either the `businessEntity` discoveryURL prefix or the `operatorName`, or both parameter values, when moving a system from a staging environment to a production environment. See ["businessEntityURLPrefix"](#) on page 10-69 and ["operatorName"](#) on page 10-76 for more information.

## Web Services Discovery

To discover Web Services in OracleAS UDDI Registry, you browse the registry using tools or using the Inquiry API. These methods are described in the following sections:

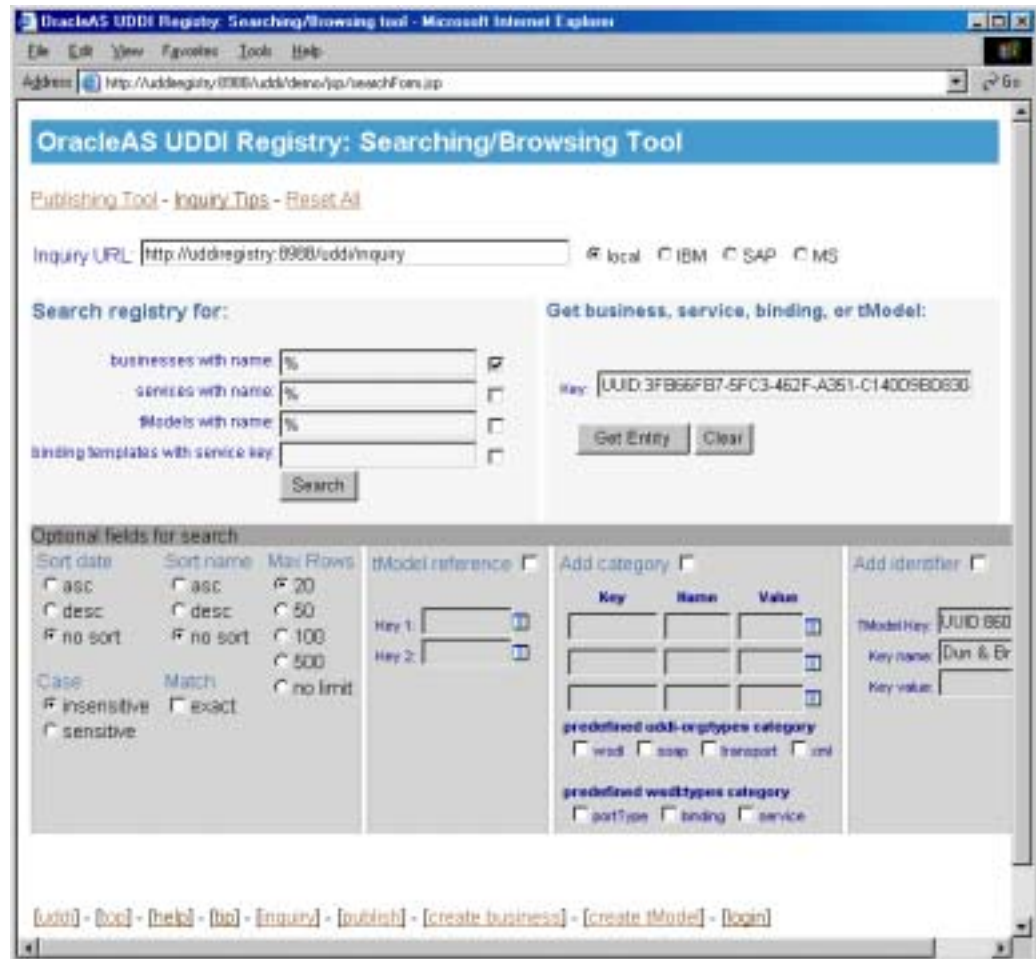
- [Using the OracleAS UDDI Registry Searching and Browsing Tool](#)
- [Using Other Tools to Discover Web Services](#)
- [Using the OracleAS UDDI Registry Inquiry API](#)

### Using the OracleAS UDDI Registry Searching and Browsing Tool

OracleAS UDDI Registry provides a Searching and Browsing Tool that lets you search a registry by businessEntity, businessService, tModel, or bindingTemplate. To access the Searching and Browsing tool, enter the following URL:

`http://OracleAS-host:OracleAS-port/uddi`

Then, click the **UDDI Inquiry/Publishing tool** link. The Searching and Browsing Tool page is displayed, as shown in the following illustration:



### Using Other Tools to Discover Web Services

As a consumer, you can use Oracle JDeveloper to browse OracleAS UDDI Registry, or you can use third-party tools to browse and drill down for information about Web

Services from OracleAS UDDI Registry, as well as from any other accessible UDDI v1.0 Web Services registry.

## Using the OracleAS UDDI Registry Inquiry API

As a consumer, you can use the Inquiry API available for Java programmers to implement your own Web Services discovery interface. The Inquiry API, part of the UDDI Client API, provides `find` (browse and drill-down) calls and `get` calls for locating and getting information in each of the five data structures shown in [Figure 10-1](#).

The Inquiry API is a Java-based API and is provided as a convenience for Java programmers. However, programs can be written in any language and can use SOAP to discover Web Services.

The URL for the OracleAS UDDI Registry inquiry API is:

```
http://OracleAS-http-server-hostname:OracleAS-port/uddi/inquiry
```

In the URL, *OracleAS-http-server-hostname* is the host where Oracle HTTP Server is installed, and *OracleAS-port* is the port number for Oracle HTTP Server.

The Inquiry API is located in the Oracle Application Server installation directory, `${ORACLE_Home}/uddi/` for UNIX and `%ORACLE_Home_ORACLE%\uddi\` for Windows. For reference documentation for the Inquiry API, see the *Oracle Application Server Web Services UDDI Client API Reference*.

A set of sample demonstration files (`uddidemo.zip`) are located on the Oracle Technology Network (OTN) Web site at:

```
http://www.oracle.com/technology/tech/webservices/htdocs/uddi
```

The `uddidemo.zip` file contains a Java program file, `UddiInquiryExample.java`, which provides Java programmers with a starting point that demonstrates the key constructs and the sequence in using the OracleAS UDDI Registry client library.

The program example does the following:

- Gets an implementation of the `SoapTransportLiaison` interface. This is an implementation that handles the details of communication between the UDDI client and server, using SOAP and some underlying transport protocol (in this case HTTP).

```
SoapTransportLiaison transport = new OracleSoapHttpTransportLiaison();
```

- Calls a helper method to set up proxy information, if necessary. You can specify HTTP proxy information for accessing OracleAS UDDI Registry on the command line, using parameters, such as `-Dhttp.proxyHost=hostname` and `-Dhttp.proxyPort=portnum`.

```
setHttpProxy((SoapHttpTransportLiaison)transport);
```

- Uses the `SoapTransportLiaison` instance and the URL of a UDDI inquiry registry to initialize an instance of `UddiClient`, which connects to the specified OracleAS UDDI Registry. The `UddiClient` instance is the primary interface by which clients send requests to OracleAS UDDI Registry.

```
UddiClient uddiClient = new UddiClient(szInquiryUrl, null, transport);
```

---

**Note:** The `UddiClient` instance, by default, operates as a UDDI v2 client (the latest version supported). If a specific version is needed, the version can be specified either through another constructor, or the JVM property `oracle.uddi.client.defaultVersion`.

For example:

```
-Doracle.uddi.client.defaultVersion=1
```

---

- Uses the `UddiClient` instance to perform a *find business* request. Specifically, it finds all `businessEntities` that start with the letter *T* and prints out the response. Note that input parameters and return values are objects that precisely mimic the XML elements defined in the UDDI specification.

```
// Find a business with a name that starts with "T"
String szBizToFind = "T";
System.out.println("\nListing businesses starting with " + szBizToFind);
// Actual find business operation:
// First null means no specialized FindQualifier.
// Second null means no max number of entries in response.
// (For example, maxRows attribute is absent.)
BusinessList bl = uddiClient.findBusiness(szBizToFind, null, null);
// Print the response.
System.out.println("The response is: ");
List listBusinessInfo = bl.getBusinessInfos().getUddiElementList();
for (int i = 0; i < listBusinessInfo.size(); i++) {
    BusinessInfo businessInfo = (BusinessInfo)listBusinessInfo.get(i);
    System.out.println(businessInfo.getBusinessKey());
    Name name = businessInfo.getFirstNameAsName();
    if (name != null) {
        System.out.println("name=" + name.getContent() +
            " ; xml:lang=" + name.getLang());
    }

    Description description =
        businessInfo.getFirstDescriptionAsDescription();
    if (description != null) {
        System.out.println("description=" + description.getContent()
            + " ; xml:lang=" + description.getLang());
    }
}
```

- Uses the `UddiClient` instance to get a `UddiElementFactory` instance. This factory should always be used to create any UDDI objects needed for inquiries.

```
UddiElementFactory uddiEltFactory = uddiClient.getUddiElementFactory();
```

- Uses the `UddiElementFactory` instance to create a `CategoryBag` instance and its `KeyedReference`, which will be used for searching.

```
CategoryBag cb = (CategoryBag)uddiEltFactory.createCategoryBag();
KeyedReference kr =
    (KeyedReference)uddiEltFactory.createKeyedReference();
kr.setTModelKey(szCategoryTModelKey);
kr.setKeyValue(szCategoryKeyValue);
kr.setKeyName("");
cb.addUddiElement(kr);
```

- Uses the `UddiClient` instance to perform a *find service* request. Specifically, it finds a maximum of 30 services, which are classified as application service providers

(code 81.11.21.06.00) under the UNSPSC classification in any businessEntity (no businessKey is specified).

```
ServiceList serviceList =
    uddiClient.findService("", cb, null, new Integer(30));
```

- Uses the UddiElementFactory instance to retrieve an XmlWriter object. To view the raw XML data represented by an object, which extends UddiElement, *marshall* the element content to the writer, and then flush and close the writer.

```
XmlWriter writerXmlWriter =
    uddiEltFactory.createWriterXmlWriter(new PrintWriter(System.out));
serviceList.marshall(writerXmlWriter);
writerXmlWriter.flush();
```

- Finds tModel operations with multiple arguments. This is a UDDI v2 feature. A *find\_xx* request now allows multiple arguments. For example, find tModel operations that have a name pattern, such as "uddi%inquiry%" and are classified as wsdlSpec or xmlSpec in uddi-org:types taxonomy:

```
System.out.println("\nListing tModels with the name pattern \"uddi%inquiry%\"
");
System.out.println("and classified as \"wsdlSpec\" or \"xmlSpec\" ");
System.out.println("under uddi-org:types taxonomy.");
// Use UddiElement factory to create UDDI-specific objects
// that are needed in inquiries.
CategoryBag cbTM = (CategoryBag)uddiEltFactory.createCategoryBag();
KeyedReference krTM1 =
    (KeyedReference)uddiEltFactory.createKeyedReference();
krTM1.setTModelKey(CoreTModelConstants.TAXONOMY_KEY_UDDI_TYPE);
krTM1.setKeyValue(CoreTModelConstants.UDDI_TYPE_VALUE_WSDL_SPEC);
cbTM.addUddiElement(krTM1);

KeyedReference krTM2 =
    (KeyedReference)uddiEltFactory.createKeyedReference();
krTM2.setTModelKey(CoreTModelConstants.TAXONOMY_KEY_UDDI_TYPE);
krTM2.setKeyValue(CoreTModelConstants.UDDI_TYPE_VALUE_XML_SPEC);
cbTM.addUddiElement(krTM2);

FindQualifiers fqTM = (FindQualifiers)uddiEltFactory.createFindQualifiers();
List listFQTM = uddiEltFactory.createList();
listFQTM.add(FindQualifiers.OR_ALL_KEYS);
fqTM.setFindQualifierStringList(listFQTM);

// Actual find tModel operation:
// Integer(10) means a maximum of 10 tModel operations are to be returned.
TModelList tModelList = uddiClient.findTModel("uddi%inquiry%",
    null,
    cbTM,
    fqTM,
    new Integer(10));

// Print some response information.
System.out.println("The response is: ");
List listTModelInfo = tModelList.getTModelInfos().getUddiElementList();
for (int i = 0; i < listTModelInfo.size(); i++) {
    TModelInfo tModelInfo = (TModelInfo)listTModelInfo.get(i);
    System.out.println(tModelInfo.getTModelKey());
    System.out.println("name=" + tModelInfo.getName());
}
```



- Closes the UddiClient instance when finished, to release resources.  
`uddiClient.close();`
- Provides URLs (in comments) to OracleAS UDDI Registry and four public UDDI registries.

## Web Services Publishing

Web Services are published in OracleAS UDDI Registry by using one of the following interfaces:

- Oracle Enterprise Manager 10g. See "[Using Oracle Enterprise Manager for Web Services Publishing](#)" on page 10-13.
- The OracleAS UDDI Registry Publishing Tool. See "[Using the OracleAS UDDI Registry Publishing Tool](#)" on page 10-19.
- The OracleAS UDDI Registry Publishing API. See "[Using the OracleAS UDDI Registry Publishing API](#)" on page 10-28.

## Using Oracle Enterprise Manager for Web Services Publishing

Using Oracle Enterprise Manager 10g, administrators can publish Web Services in OracleAS UDDI Registry and can update discovered Web Services:

- Using the Deploy Application Wizard. The Deploy Application wizard takes you through the process of deploying a J2EE application on the OC4J container.  
See "[Publishing Web Services Using the Deploy Application Wizard](#)" on page 10-13.
- Using the UDDI Registry page. The UDDI Registry page lets you discover published Web Services and update those published Web Services.  
See "[Updating Published Web Services in OracleAS UDDI Registry](#)" on page 10-16.

---

**Note:** To use any of the UDDI functions in Oracle Enterprise Manager, you must have initialized the UDDI registry by pingging the UDDI inquiry servlet end point. See "[Configuring OracleAS UDDI Registry](#)" on page 10-6 for more information.

If you have accessed any of the UDDI-related pages before you initialize the UDDI registry, you must restart Oracle Enterprise Manager.

---

### Publishing Web Services Using the Deploy Application Wizard

As an administrator, you can publish J2EE Web Services, which are produced by the OracleAS Web Services assembly tool, using the Oracle Enterprise Manager Deploy Application wizard.

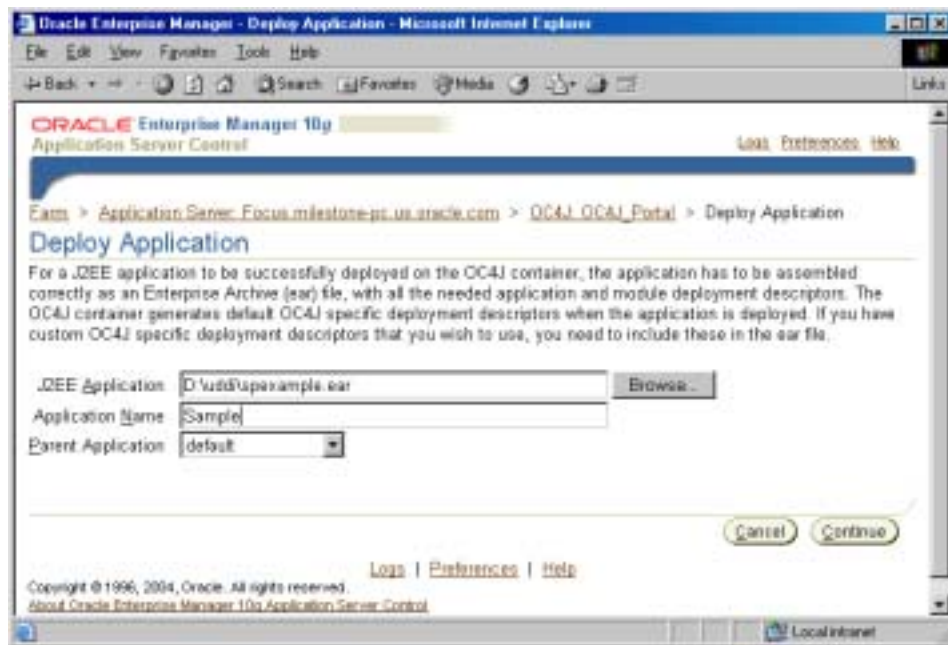
To publish J2EE Web Services, you must first assemble them as J2EE Enterprise Archive (EAR) files. See [Chapter 9](#) for more information. See the *Oracle Application Server Containers for J2EE User's Guide* for information about EAR file-based deployment of J2EE Web applications.

You can use the wizard to publish Web Services servlets that are found in an EAR file. Any Web Services servlet in an application that you want to access must be published to OracleAS UDDI Registry to one or more desired categories within one or more of

the classifications provided. Any unpublished Web Services servlet in an application appears with the status of `Not Published`. When the Web Services servlet is published, the status changes to `Published`.

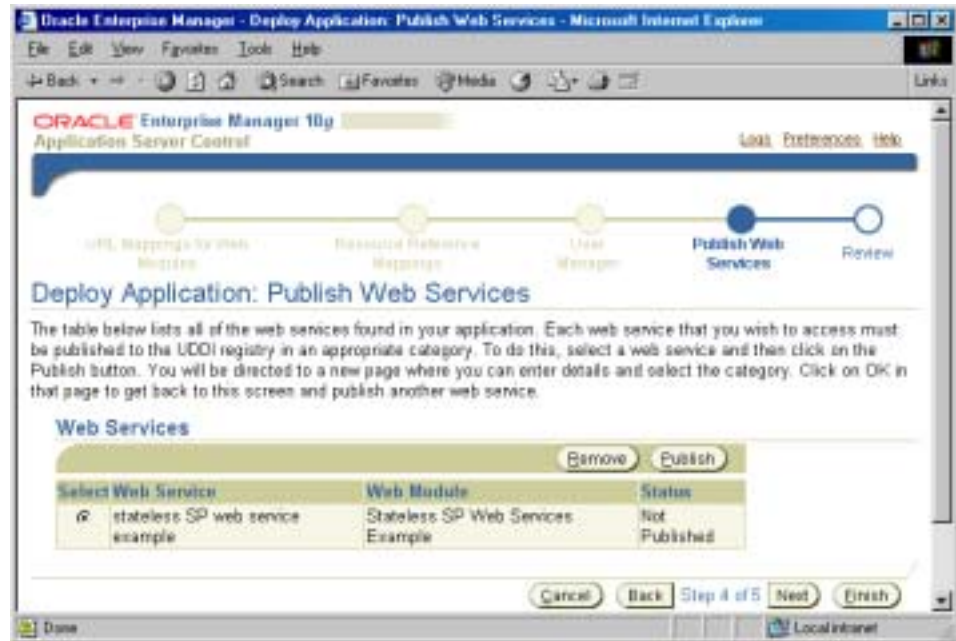
After you have initialized the OracleAS UDDI Registry, take the following steps:

1. Invoke Oracle Enterprise Manager and navigate to the Application Server *Instance-name* page. From the **System Components** table, select an OC4J instance. By default, the UDDI registry is deployed in the OC4J\_PORTAL instance.
2. In the OC4J:*oc4j-name* page, click **Applications**.
3. In the **Deployed Applications** section, click **Deploy Ear File** to invoke the **Deploy Application** wizard.
4. In the **Deploy Application** wizard, in the Deploy Application page, specify the ear file location in the **J2EE Application** field and the application name in the **Application Name** field, as shown in the following illustration:



Click **Continue**.

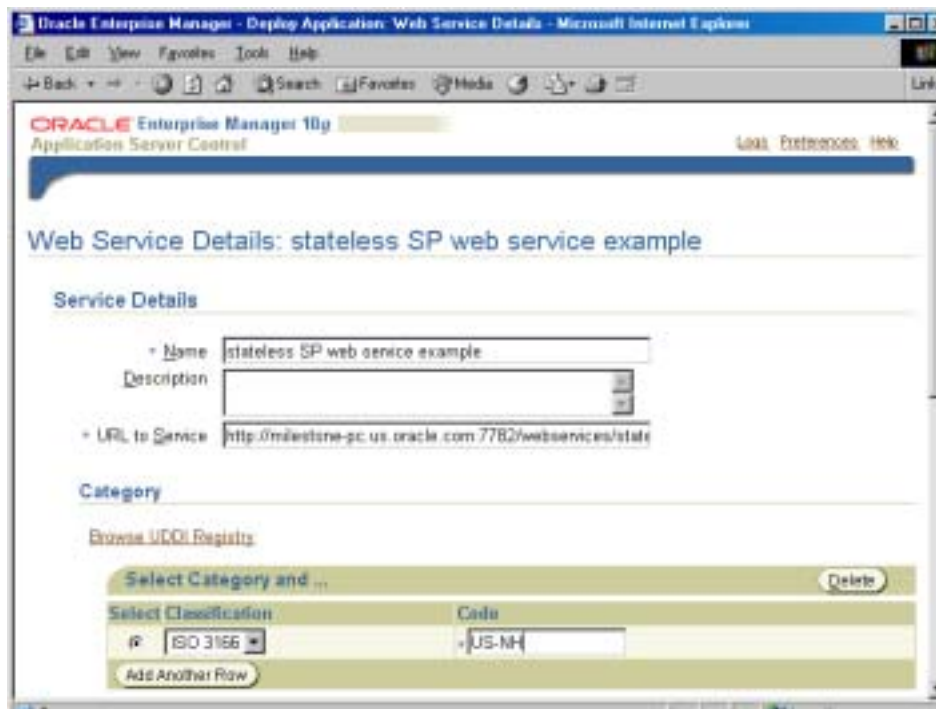
5. For the next pages, take the defaults until you reach the Publish Web Services page, shown in the following illustration:



If the wizard does not display the Publish Web Services page, either the UDDI registry has not been initialized or the OracleAS Infrastructure is not running. See ["Configuring OracleAS UDDI Registry"](#) on page 10-6 for information about initializing the UDDI registry. Then, you may need to restart Oracle Enterprise Manager.

6. At the Publish Web Services page, select the desired Web Services to register from the list of Web Services known to the application whose **Status** is **Not Published**. Then, click **Publish** to continue to the Web Service Details page.
7. At the Web Service Detail page, review, edit, or enter the information as needed in each of the fields in the **Service Details** and **tModel Details** sections. OracleAS UDDI Registry automatically adds the **Name** and **URL to Service**.
  - To specify a category, select a **Classification** and **Code** in the Category Section.
  - To specify categories to which the Service or tModel are to be registered, in the **Service Details** or the **tModel Details** sections, click **Browse UDDI Registry**. Then, browse to the desired classification, and drill down as needed through each desired category, noting all desired category names and values.
  - To add an empty row of category information, click **Add Another Row**. Select the desired classification, then enter the value code and its corresponding category name for the desired category.

The following illustration shows the top part of the Web Service Details page:



8. After entering all the required information on the Web Service Details page, publish the Web Services to OracleAS UDDI Registry by clicking **OK**. You return to the Publish Web Services page.
9. In the Publish Web Services page, select another Web Service to publish and repeat this entire process again as described in Step 6 and Step 7.
10. After publishing all Web Services for this application, click **Next** to continue to the Review page where you can review the application deployment information.
11. If there are no further changes, click **Deploy** to deploy the J2EE application on the OC4J container. Doing this returns you to the Oracle Enterprise Manager OC4J Home page.

After deployment, metadata describing the Web Services that you chose to publish has been added to OracleAS UDDI Registry.

### Updating Published Web Services in OracleAS UDDI Registry

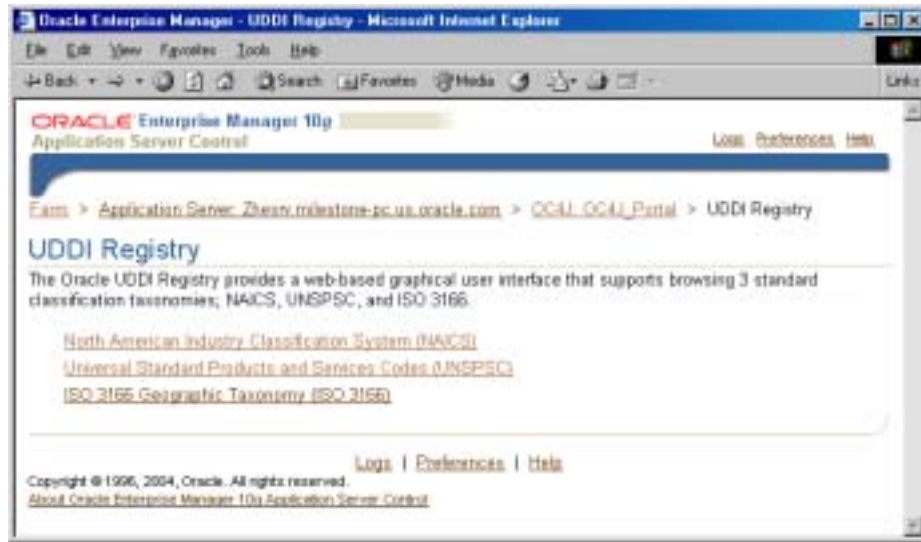
You can use Oracle Enterprise Manager to browse, drill down, and get information about Web Services published for categories in OracleAS UDDI Registry. You can update the discovered published Web Services.

To update published Web Services using Oracle Enterprise Manager 10g, do the following:

1. Invoke Oracle Enterprise Manager and navigate to the Application Server *Instance-name* page. From the **System Components** table, select an OC4J instance.
2. In the OC4J:*oc4j-name* page, click the **Administration** link. On the Administration page, click the **UDDI Registry** link in the **Related Links** section.

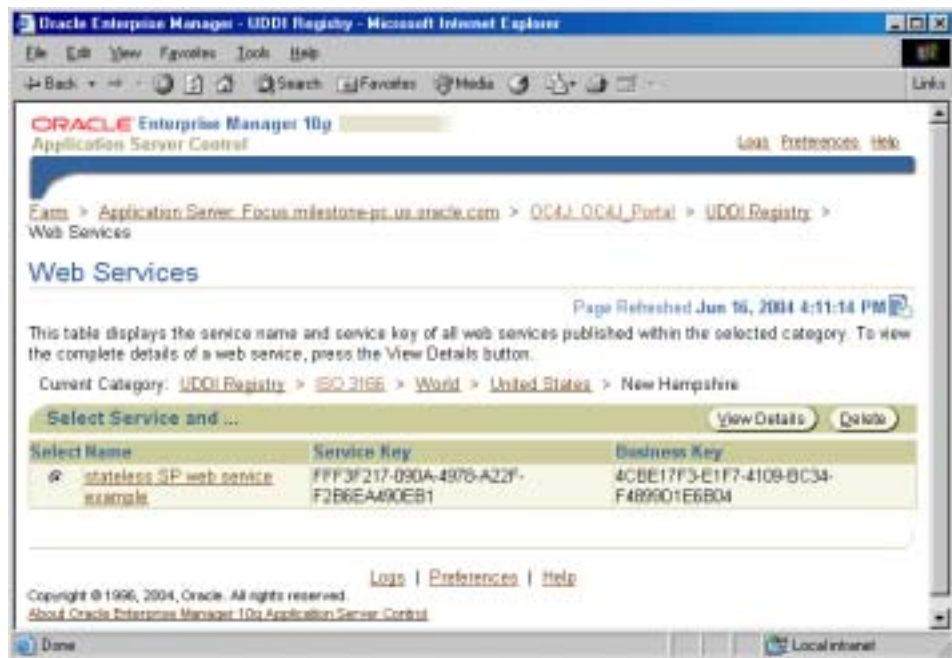
If **UDDI Registry** is not listed in the **Related Links** column, initialize the OracleAS UDDI Registry by pinging the UDDI inquiry servlet end point (see "[Configuring OracleAS UDDI Registry](#)" on page 10-6.) Then, restart Oracle Enterprise Manager.

- In the UDDI Registry page, select one of the three standard classifications, NAICS, UNSPSC, or ISO-3166, by clicking its link. The following illustration shows the UDDI Registry page:



- In the UDDI Registry: *Classification\_Name* page, you can drill down from category to subcategory to discover published Web Services associated with any category or subcategory. Each classification is organized in a hierarchical tree. Navigate down a particular branch by clicking the category name to determine all its subcategory names, and so forth. As you navigate down a branch, also note the change in the category code value.

The following illustration shows the page:

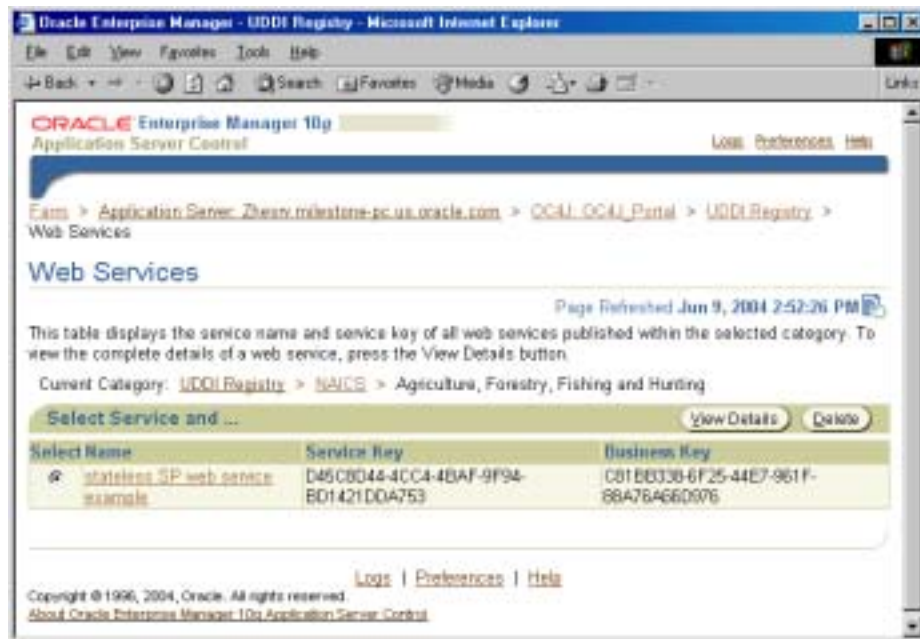


Navigate to the desired category or subcategory by successively clicking the desired categories.

5. To view all Web Services published in a particular category, select the corresponding radio button in the **Select** column for that category, and click **View Details**.

The Web Services page lists all Web Services published for that category name. For Web Services listed for the selected category, the corresponding service name, service key, and business key are also listed. If the selected category or subcategory has no published Web services, none is listed.

The following illustration shows the Web Services page:



6. To view the complete details of a particular published Web Services listed for a category, either click its service name or select its corresponding radio button in the **Select** column and click **View Details**.

Click the desired service name.

7. The Web Service Details page displays detailed information for the selected Web Service published in OracleAS UDDI Registry. This information includes:
  - **Service Details:** Information such as the Web Services name, Web Services description, and the URL of the Web Services access point.  
Service Category: The classification and the corresponding code value and its category name.
  - **tModel Details:** Information that describes the interface that the Web Services implements, such as the tModel name, tModel description, and URL to the interface specification, typically a WSDL document.  
tModel Category: The classification and the corresponding code value and its category name.

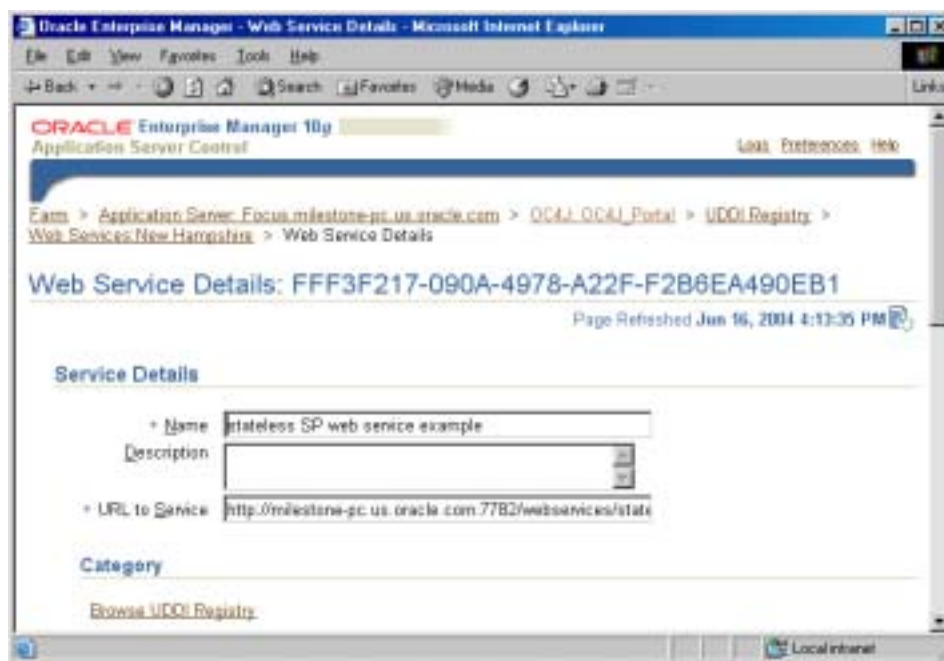
On this page, you can:

- Browse OracleAS UDDI Registry to look for categories in which to register Web Services: Click **Browse UDDI Registry**.
- Add categories to which either the Web Services or tModel are to be registered: Click **Add Another Row**.

- Remove categories to which the Web Services and tModel are registered: Click **Delete**.
- 8. After making all selections or completing all changes for this Web Services, click **Apply** to save your changes.

If you have made changes to any field and you decide you want to return to the original set of values for all selections, click **Revert**. The page refreshes with the original set of values for all selections as if you had just begun your current session.

To update other published Web Services for the same category, at the top of the **Web Service Details** page, select the **Web Services: *Classification\_Name*** link to return to the desired Web Services:*Classification\_Name* page. The following illustration shows the Web Service Details page:



When you return to the desired Web Services:*Classification\_Name* page, select another Web Service to view in more detail, make any necessary changes, and finally click **Apply** to save your changes.

Alternatively, you can click **Browse UDDI Registry** to return to the UDDI Registry page, where you can navigate to another classification to discover Web Services for other categories. At each desired category, select the desired Web Services to view the details, make any necessary changes, and finally click **Apply** to save your changes.

## Using the OracleAS UDDI Registry Publishing Tool

OracleAS UDDI Registry provides a Publishing tool that enables you to create a new businessEntity, containing new businessServices and bindingTemplates, or tModel. To access the Publishing tool, enter the following URL in a browser:

```
http://OracleAS-host:OracleAS-port/uddi
```

The OracleAS UDDI Registry page is displayed.

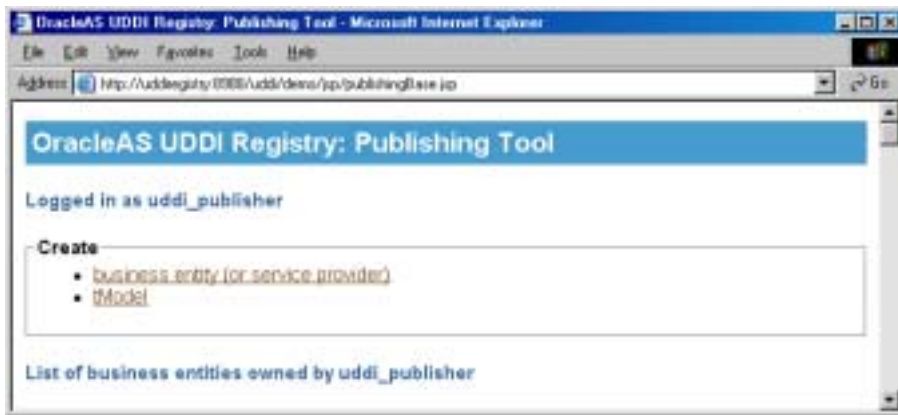
This section uses an example that publishes a service for a Google-based search. It shows how to publish to the UDDI registry including how to create a tModel that is

mapped to a WSDL interface specification and how to create a `businessEntity` and a `businessService`.

Take the following steps:

1. Click the **UDDI Inquiry/Publishing tool** link. Then, on the OracleAS UDDI Registry: Searching and Browsing Tool page, click **Publishing Tool**.
2. On the Log Into Publishing Service page, enter the user name and password for the `uddi_publisher` user and click **Login**. (See [Table 10-2](#) for default passwords.)

The Publishing Tool page is displayed, as shown in the following figure:



3. In the Publishing Tool page, you can create:
  - A new `businessEntity`
  - A new `tModel`

The first step in registering a Web service in OracleAS UDDI Registry is to create a `tModel`. This example creates a `tModel` that represents a Google-compatible service.

To create a new `tModel`, click **tModel**. The Publish `tModel` page is displayed.

4. In the **Basic information** section, specify the following information:
  - a. For **Name**, specify a name so that this `tModel` can be located. For example, enter `urn:google.com:search-interface`.
  - b. For **Description**, specify a description of the `tModel`, such as **Google search interface**.
  - c. For **Overview Document URL**, specify the URL to the interface specification, typically a WSDL document. In this case, Google provides a WSDL document, <http://api.google.com/GoogleSearch.wsdl>.
  - d. For **Overview Document Description**, enter a description of the overview document.
5. In the **Category** section, you categorize the `tModel`, in this case as a WSDL-based interface. To find information about the `tModel` key and key name and value, click the **browse category** icon at the end of the first row. The Category Browsing page is displayed.

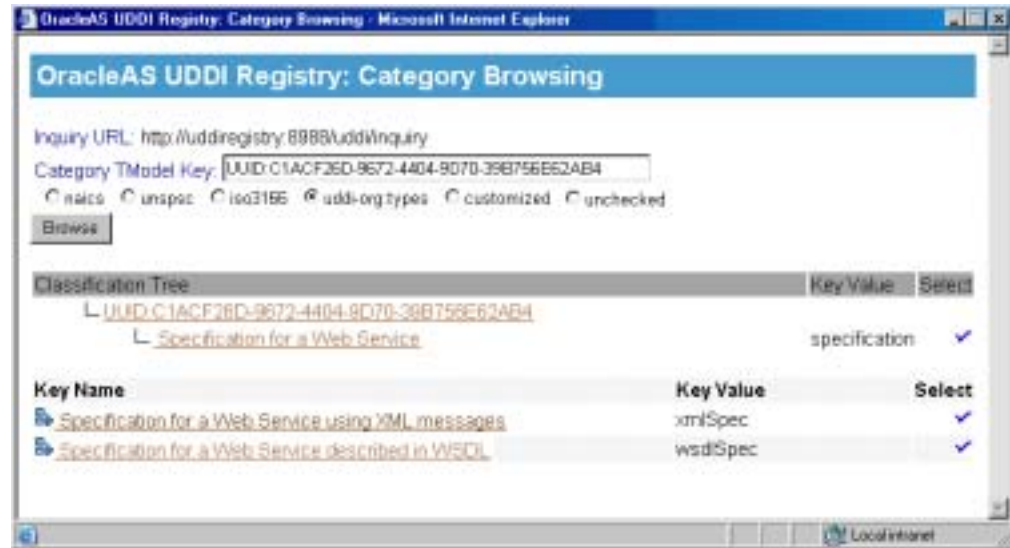
Take the following steps:

- a. Select `uddi-org:types` and click **Browse**.

The Classification Tree for this selection is displayed.



- b. Click **Specification for a Web Service**. The Classification Tree for this selection is displayed, as shown in the following figure:

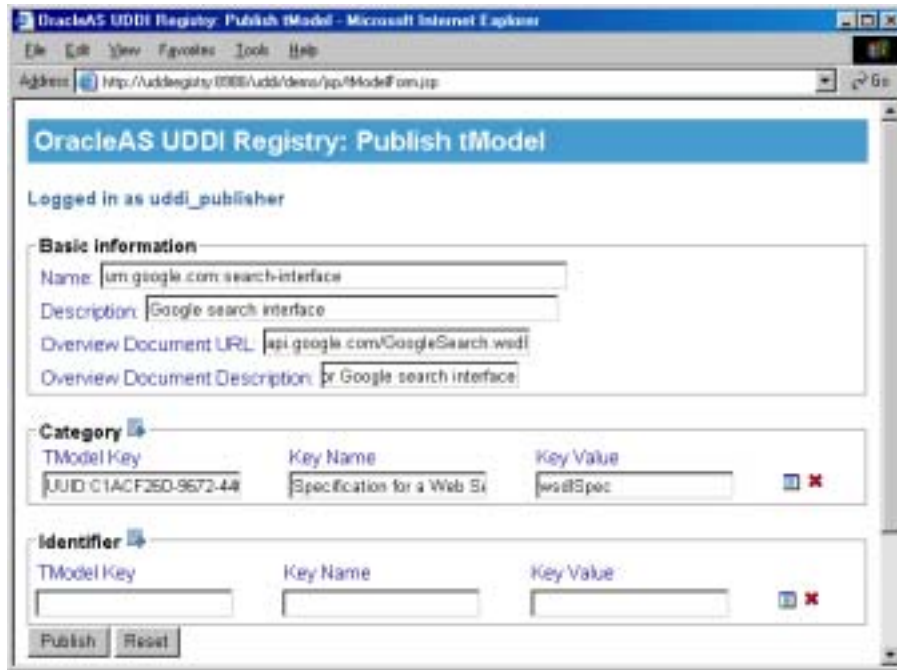


- c. In the row **Specification for a Web Service described in WSDL**, click the **Select** icon.

In the Publish tModel page, the information you selected is automatically entered into the **Category** section. Now, it contains the following information:

- For **tModel Key**: **uuid:C1ACF26D-9672-4404-9D70-39B756E62AB4**
  - For **Key Name**: **Specification for a Web Service described in WSDL**
  - For **Key Value**: **wsdlSpec**
6. The **Identifier** section specifies identifying information, such as from the D-U-N-S or Thomas Register systems (see "[Support for Standard Classification and Identifier Systems](#)" on page 10-4 for more information). For this example, leave this section blank.

The following figure shows the Publish tModel page:



7. Click the **Publish** button to publish the tModel. The TModel Details page is displayed, as shown in the following figure:



Note that the key (UUID:B960F57E-54BF-4DB8-BC36-F2802FE6BEF0) for this tModel is randomly assigned by the registry.

8. Click the **Publish** link at the bottom of the page to return to the Publishing Tool page.
9. Create a businessEntity, which contains details about the business, such as the name, contacts, and categories with which the businessEntity is associated. Click **business entity (or service provider)**. The Publish Business Entity (Service Provider) page is displayed.

If a businessEntity already exists, you do not need to create a new entity. In the Publishing Tool page, click the link to the existing businessEntity. Then, proceed to Step 15.

10. In the **Business Details** section of the Publish Business Entity (Service Provider) page, specify the following information:
  - a. For **Name**, specify a name so that this business can be located. For example, enter **Google Service Provider**.
  - b. For **Description**, specify a description of the business.
11. In the **Contacts** section, enter information that you want to appear in the UDDI registry. Specify information for the following fields:
  - a. For **name**, specify the name of a contact.
  - b. For **voice**, specify a voice mail number.
  - c. For **email**, specify an email address.
  - d. For **address**, specify a mailing address.
12. In the **Category** section, add any categories with which the business should be associated. For this example, use the ISO-3166 geographic taxonomy to provide geographic categorization of the business.

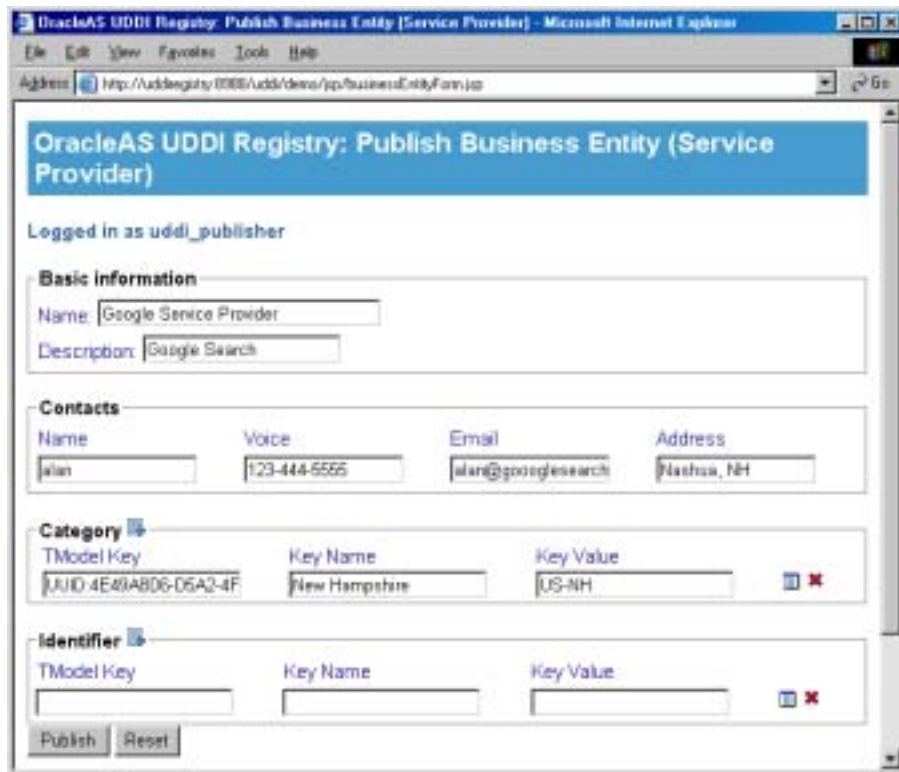
To find information about the tModel key and key name and value of the category, click the **browse category** icon at the end of the first row. The Category Browsing page is displayed. Then, take the following steps:

  - a. Select **iso3166** and click **Browse**. The Classification Tree for this selection is displayed.
  - b. Drill down to select the geographic category. For this example, click **World**, click **United States**, and then click the select icon for **New Hampshire**.

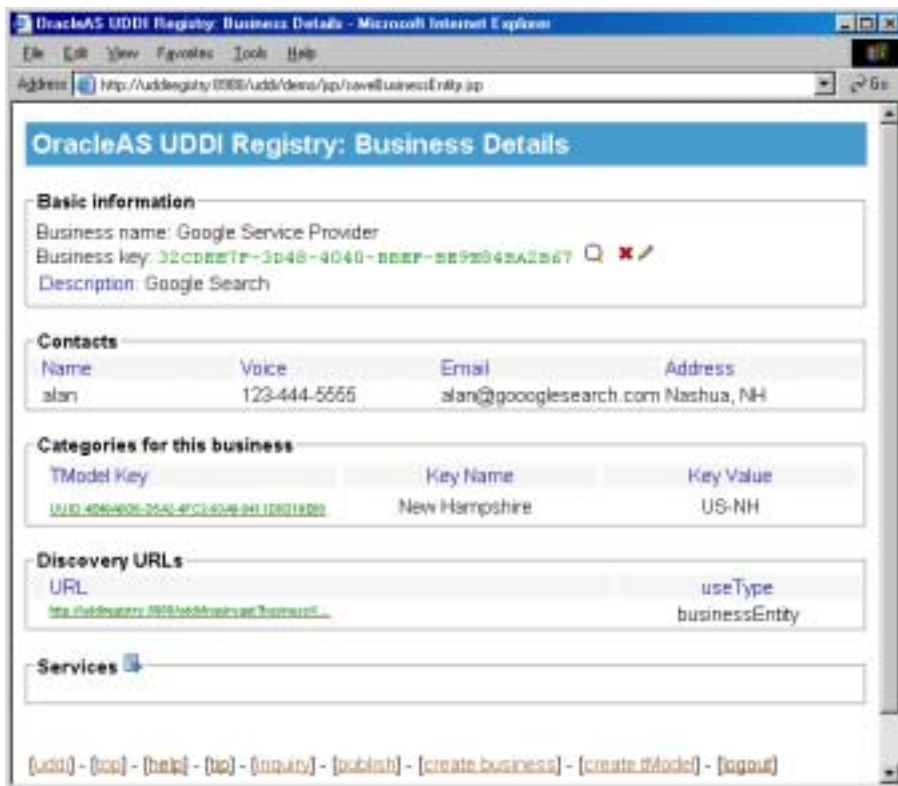
In the Publish Business Entity page, the information you selected is automatically entered into the **Category** section. Now, it contains the following information:

  - For **tModel Key**: **uuid:4E49A8D6-D5A2-4FC2-93A0-0411D8D19E88**
  - For **Key Name**: **New Hampshire**.
  - For **Key Value**: **US-NH**.
13. The information for the **Identifier** section is optional. For this example, leave it blank.

The following figure shows the Publish Business Entity (Service Provider) page:



14. Click **Publish** to publish the business. The Business Details page is displayed:



Note that the Business Key and the Discovery URL are generated by OracleAS UDDI Registry. With the Discovery URL, you can retrieve the businessEntity using the HTTP Get method.

15. In the **Services** section of the Business Details page, click the icon to the right of **Services** to enter information about the businessServices offered by the businessEntity and the categorization of the service. The Publish Business Service page is displayed.
16. In the **Basic information** section, specify the following information:
  - a. The key for the business is automatically entered in **Owning business**.
  - b. For **Name**, enter a name for the businessService.
  - c. For **Description**, specify a description of the businessService.
17. In the **Category** section, add any categories to detail the intended use of this service. For this example, add a category from the NAICS taxonomy to specify that that category is an online search tool.

To find information about the tModel key and key name and value of the category, click the **browse category** icon at the end of the first row. The Category Browsing page is displayed. Then, take the following steps:

- a. Select **naics** and click **Browse**. The Classification Tree for this selection is displayed.
- b. Drill down to select the category. Click **Information**, then **Information Services and Data Processing Services**, then **Information Services**, and **Other Information Services**. Then, click the **Select** icon for **On-Line Information Services**.

In the Publish Business Service page, the information you selected is automatically entered into the **Category** section. Now, it contains the following information:

- For **tModel Key**: **uuid:C0B9FE13-179F-413D-8A5B-5004DB8E5BB2**
- For **Key Name**: **On-Line Information Services**
- For **Key Value**: **514191**

The following figure shows the Publish Business Service page:



18. Click the **Publish** button to publish the service. The Service Details page is displayed, as shown in the following figure:



19. Create a bindingTemplate, which contains details of how and where the service provided by this businessService is accessed. On the Service Details page, click the icon to the right of **Binding Details**.

20. In the Publish Binding Template page, OracleAS UDDI Registry automatically fills in the field **Owning Service Key** with the key for the service you just created. In the **Basic information** section, enter the following information:

- For **Description**, specify a description for the bindingTemplate.
- For **Access Point**, specify the access point for accessing the service. In this case, review the source code for `http://api.google.com/GoogleSearch.wsdl` to find the access point. The WSDL file contains the following:

```
<soap:address location="http://api.google.com/search/beta2"/>
```

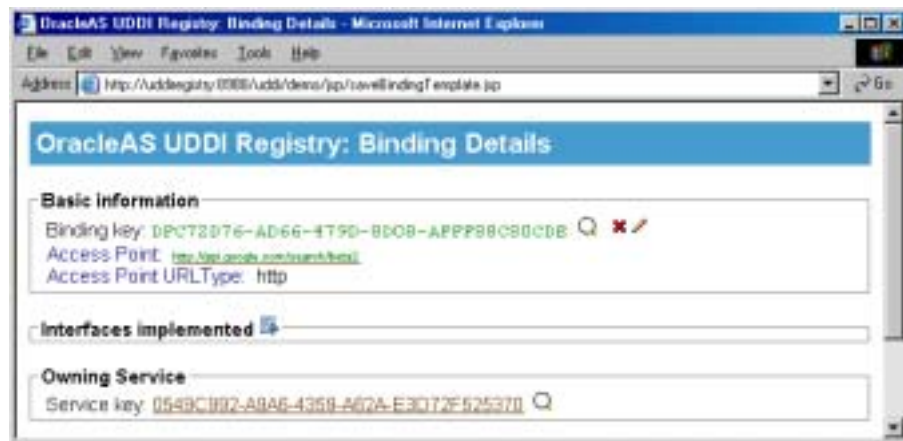
Enter `http://api.google.com/search/beta2` as the Access Point.

- For **Access Point URL Type**, select **HTTP**.

The following figure shows the Publish Binding Template page:



21. Click **Publish** to publish the bindingTemplate. The Binding Details page is displayed, as shown in the following figure:



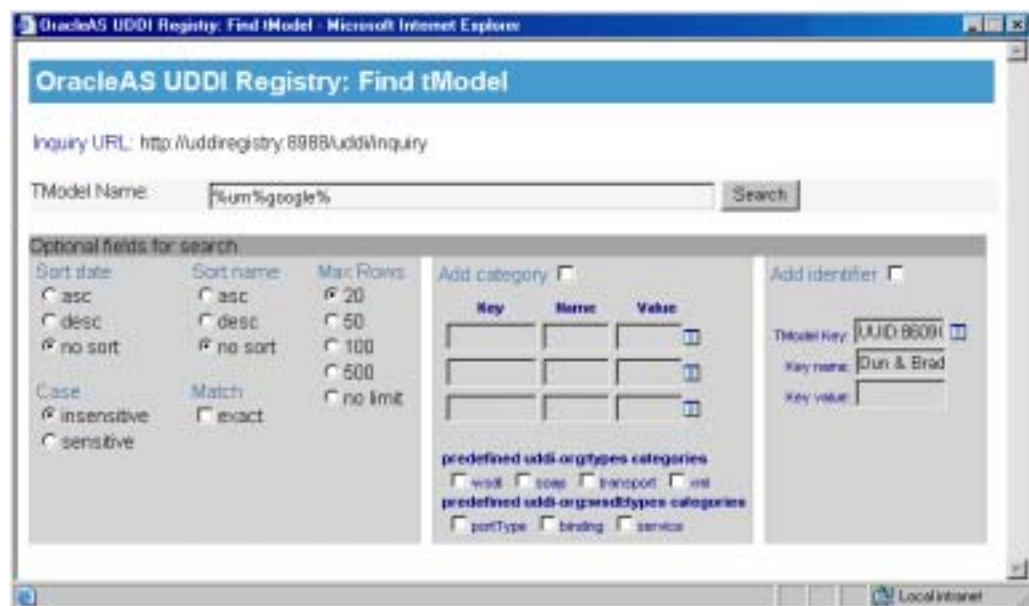
22. Click the icon to the right of **Interfaces implemented** to add a new interface reference.

The Publish Interface Reference page is displayed.

23. On this page, OracleAS UDDI Registry automatically fills in the **Binding Key** field. Click the **Search a suitable tModel** icon, which is at the end of the Interface (tModel) Key line.

The Find tModel page is displayed.

24. For **tModel Name**, enter the beginning of the name of the tModel you created, for example, **%urn%google%**. The following figure shows the Find tModel page:

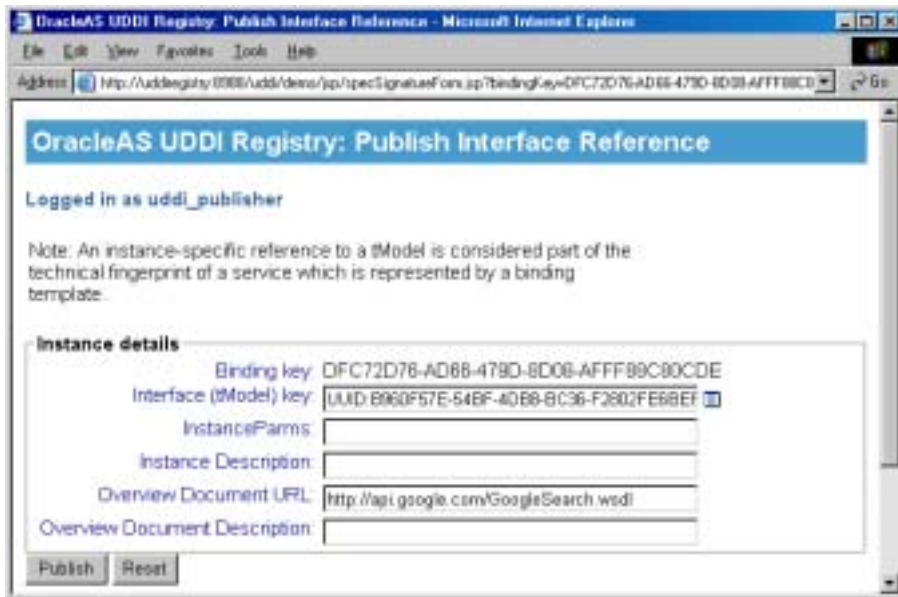


25. Click **Search**.

26. The tModel you created earlier is returned. Click **Select**.

The Publisher Interface Reference page is displayed. The tModel Key is automatically entered in the **Interface (tModel) key** field of the page.

27. For **Overview Document URL**, enter the URL for the interface specification. In this case, enter **http://api.google.com/GoogleSearch.wsdl**, as shown in the following figure:



28. Click the **Publish** button. The Binding Details page is displayed. As the following figure shows, it now contains information about the implemented interfaces:



## Using the OracleAS UDDI Registry Publishing API

The OracleAS UDDI Registry Publishing API lets consumers (programmers), following authentication, publish Web Services by providing save and delete calls for each of the five key UDDI data structures (businessEntity, businessService, bindingTemplate, tModel, and publisherAssertion).



The Publishing API, part of the UDDI Client API, allows programmers to publish Web Services using the Java language. Programs can be written in any language, using SOAP to publish Web Services. The Java API is provided as a convenience for Java programmers.

The Publishing API is located in the Oracle Application Server installation directory, `${ORACLE_Home}/uddi/` for UNIX and `%ORACLE_Home_ORACLE%\uddi\` for Windows. The API documentation can be found in the *Oracle Application Server Web Services UDDI Client API Reference Javadoc*.

A set of sample demonstration files (`uddidemo.zip`) are located on the Oracle Technology Network (OTN) Web site at:

<http://www.oracle.com/technology/tech/webservices/htdocs/uddi>

### UddiPublishingExample.java Example

The `uddidemo.zip` file contains a Java program file, `UddiPublishingExample.java`, that provides Java programmers with a starting point that demonstrates the key constructs and the sequence in using the OracleAS UDDI Registry client library.

The program example does the following:

- Gets an instance of `SoapHTTPTransportLiaison`. This is an implementation that handles the details of communication between the UDDI client and server using SOAP and some underlying transport protocol (in this case HTTP).

```
SoapHttpTransportLiaison transport = new OracleSoapHttpTransportLiaison();
```

- Sets the proxy information for the transport if the system properties `http.proxyHost` and `http.proxyPort` are set. These properties can be set on the command line. If these properties are not set, this command has no effect.

```
setHttpProxy((SoapHttpTransportLiaison)transport);
```

- Uses a `SoapTransportLiaison` instance and the URL of a UDDI publishing registry to initialize an instance of `UddiClient`, which connects to the specified OracleAS UDDI Registry. The `UddiClient` instance is the primary interface by which clients send requests to OracleAS UDDI Registry. Authentication is done using the UDDI `get_authToken` message in this example.

```
SimpleAuthenticationLiaison auth =
    new SimpleAuthenticationLiaison(szUserName, szPassword);
UddiClient uddiClient = new UddiClient(null, szPublishingUrl,
    transport, auth);
```

---

**Note:** The `UddiClient` instance, by default, operates as a UDDI v2 client (the latest release supported). If a specific release is needed, the release can be specified, either through another constructor, or by the JVM property `oracle.uddi.client.defaultVersion`.

---

- Performs authentication. You should make this call before doing any publishing.

```
UddiClient.authenticate();
```

- Uses `UddiClient` to get a `UddiElementFactory` instance. This factory should always be used to create any UDDI objects needed.

```
UddiElementFactory uddiEltFactory = uddiClient.getUddiElementFactory();
```

- **Creates and includes the OverviewDoc data structure in the tModel data structure by using the UddiElementFactory instance.**

```
OverviewDoc overviewDocTm =
    (OverviewDoc)uddiEltFactory.createOverviewDoc();
tModel.setOverviewDoc(overviewDocTm);
overviewDocTm.setOverviewURL("http://api.google.com/GoogleSearch.wsdl");
```

- **Creates a tModel data structure that represents a Google-compatible service by using the UddiElementFactory instance.**

```
TModel tModel = (TModel)uddiEltFactory.createTModel();
tModel.setName("urn:google.com:search-interface");
```

- **In the tModel data structure, uses the UddiElementFactory instance to create a CategoryBag data structure and its keyedReference data structure, which will be used for searching. Classify the tModel data structure as a SOAP/WSDL-based interface and put it under the "applicable service providers" category.**

```
CategoryBag catBagTm =
    (CategoryBag)uddiEltFactory.createCategoryBag();
tModel.setCategoryBag(catBagTm);

KeyedReference krTm1 =
    (KeyedReference)uddiEltFactory.createKeyedReference();

catBagTm.addUddiElement(krTm1);
krTm1.setTModelKey(CoreTModelConstants.TAXONOMY_KEY_UDDI_TYPE);
krTm1.setKeyName("soapSpec");
krTm1.setKeyValue("soapSpec");

KeyedReference krTm2 =
    (KeyedReference)uddiEltFactory.createKeyedReference();
catBagTm.addUddiElement(krTm2);
krTm2.setTModelKey(CoreTModelConstants.TAXONOMY_KEY_UDDI_TYPE);
krTm2.setKeyName("wsdlSpec");
krTm2.setKeyValue("wsdlSpec");

KeyedReference krTm3 =
    (KeyedReference)uddiEltFactory.createKeyedReference();
catBagTm.addUddiElement(krTm3);
krTm3.setTModelKey(CoreTModelConstants.TAXONOMY_KEY_UNSPSC_7_3);
krTm3.setKeyName("application service providers");
krTm3.setKeyValue("81.11.21.06.00");
```

- **Publishes the Google search interface tModel business operation.**

```
System.out.println("\nPublish the google search interface tModel.");
TModel tMSaved = uddiClient.saveTModel(tModel);
String szGoogleTModelKey = tMSaved.getTModelKey();
System.out.println("The tModel is saved with tModelKey assigned to be " +
    szGoogleTModelKey);
```

- **Creates a businessEntity data structure that represents a Google-compatible service by using the UddiElementFactory instance.**

```
BusinessEntity businessEntity =
    (BusinessEntity)uddiEltFactory.createBusinessEntity();
businessEntity.setName("ACME search Inc.", "en");
```

In the `businessEntity` data structure, uses the `UddiElementFactory` instance to create a `CategoryBag` data structure and its `keyedReference` data structure, which will be used for searching. Classify the `businessEntity` data structure under the "information services and data processing services" category.

```
KeyedReference krBel =
    (KeyedReference)uddiEltFactory.createKeyedReference();
catBagBe.addUddiElement(krBel);
krBel.setTModelKey(CoreTModelConstants.TAXONOMY_KEY_NAICS_1997);
krBel.setKeyName("Information Services and Data Processing Services");
krBel.setKeyValue("514");
```

- Creates a `businessService` data structure that represents a Google-compatible service by using the `UddiElementFactory` instance.

```
BusinessServices businessServices =
    (BusinessServices)uddiEltFactory.createBusinessServices();
businessEntity.setBusinessServices(businessServices);
BusinessService businessService =
    (BusinessService)uddiEltFactory.createBusinessService();
businessServices.addUddiElement(businessService);
businessService.setName("ACME Web Search service", "en");
```

In the `businessService` data structure, uses the `UddiElementFactory` instance to create a `CategoryBag` data structure and its `keyedReference` data structure, which will be used for searching. Classify the `businessService` data structure under the "application service providers" category.

```
CategoryBag catBagBs =
    (CategoryBag)uddiEltFactory.createCategoryBag();
businessService.setCategoryBag(catBagBs);
KeyedReference krBs1 =
    (KeyedReference)uddiEltFactory.createKeyedReference();
catBagBs.addUddiElement(krBs1);
krBs1.setTModelKey(CoreTModelConstants.TAXONOMY_KEY_UNSPSC_7_3);
krBs1.setKeyName("application service
providers");krBs1.setKeyValue("81.11.21.06.00");
```

- Creates the `bindingTemplate` data structure that represents a Google-compatible service by using the `UddiElementFactory` instance.

```
BindingTemplates bindingTemplates =
    (BindingTemplates)uddiEltFactory.createBindingTemplates();
businessService.setBindingTemplates(bindingTemplates);
BindingTemplate bindingTemplate =
    (BindingTemplate)uddiEltFactory.createBindingTemplate();
bindingTemplates.addUddiElement(bindingTemplate);
```

- Creates and includes the access point in the `bindingTemplate` data structure by using the `UddiElementFactory` instance.

```
AccessPoint accessPoint =
    (AccessPoint)uddiEltFactory.createAccessPoint();
bindingTemplate.setAccessPoint(accessPoint);
accessPoint.setUrlType("http");
accessPoint.setContent("http://foobar.net/search-g");
```

- Creates and includes the `tModel` instance details in the `bindingTemplate` data structure by using the `UddiElementFactory` instance.

```
TModelInstanceDetails tModelInstanceDetails =
    (TModelInstanceDetails)uddiEltFactory.createTModelInstanceDetails();
```

```
bindingTemplate.setTModelInstanceDetails(tModelInstanceDetails);
```

- Declares that the `bindingTemplate` data structure implements the Google search interface.

```
TModelInstanceInfo tModelInstanceInfo =
    (TModelInstanceInfo)uddiEltFactory.createTModelInstanceInfo();
tModelInstanceDetails.addUddiElement(tModelInstanceInfo);
tModelInstanceInfo.setTModelKey(szGoogleTModelKey);
```

- Publishes the `businessEntity` data structure and its contained `businessService` and `bindingTemplate` data structures.

```
System.out.println("Publish the ACME Search Inc. businessEntity...");
BusinessEntity bESaved = uddiClient.saveBusiness(businessEntity);
System.out.println("The saved businessEntity (in XML) is:");
```

```
bESaved.setName("The ACME search Inc.", "en");
BusinessEntity bEUpdated = uddiClient.saveBusiness(bESaved);
```

- Uses the `UddiElementFactory` instance to retrieve an `XmlWriter` object. To view the raw XML data represented by an object, which extends `UddiElement`, *marshall* the element content to the writer, and then flush and close the writer.

```
XmlWriter writerXmlWriter =
    uddiEltFactory.createWriterXmlWriter(new PrintWriter(System.out));
bESaved.marshall(writerXmlWriter);
writerXmlWriter.flush();
writerXmlWriter.close();
```

- Closes the `UddiClient` instance when finished to release resources and to log out from the registry.

```
uddiClient.close();
```

### UddiPublisherAssertionExample.java Example

The `uddidemo.zip` file contains a Java program file, `UddiPublisherAssertionExample.java`, which provides Java programmers with a starting point that demonstrates the key constructs and the sequence in using the OracleAS UDDI Registry client library for publisher assertion-related operations.

A **publisherAssertion**, which is a UDDI v2 feature, is an assertion made by a publisher who is expressing a particular fact about a business registration and its relationships to other business data within OracleAS UDDI Registry. A `publisherAssertion` is used to establish visible relationships between registered data. Once completed, a set of assertions can be seen by the general inquiry message named `findRelatedBusinesses`.

The program example does the following:

- Initializes instances of two `UddiClients`.

```
UddiClient uddiClient1 = createUddiClient(szInquiryUrl,
                                         szPublishingUrl, szUserName1, szPassword1);
UddiClient uddiClient2 = createUddiClient(szInquiryUrl, szPublishingUrl,
                                         szUserName2, szPassword2);
DispositionReport dispositionReport = null;
```

- Creates the `businessEntity` data structures to be used.

```
String bEKey1 = createBusinessEntity(uddiClient1,
                                     "bE1 - UddiPublisherAssertionExample");
```

```
String bEKey2 = createBusinessEntity(uddiClient2,
                                     "bE2 - UddiPublisherAssertionExample");
```

- **Creates, for uddiClient1, a publisherAssertion that represents a peer-to-peer relationship from bE1 to bE2.**

```
System.out.println("");
System.out.println("uddiClient1 attempts to create a peer-to-peer relationship
");
System.out.println("from bE1 to bE2...");
dispositionReport = uddiClient1.addPublisherAssertion
    (createPeerToPeerPublisherAssertion(uddiClient1, bEKey1, bEKey2));
System.out.println("Done.");
```

- **Makes a query for uddiClient1 for relationships yet to be established; that is, looking for those relationships that the toKey side has not yet acknowledged.**

```
AssertionStatusReport assertionStatusReport1 =
    uddiClient1.getAssertionStatusReport
        (AssertionStatusItem.COMPLETION_STATUS_TOKEY_INCOMPLETE);
printOutXml("pending relationships for uddiClient1: case toKey incomplete",
            assertionStatusReport1);
```

- **Makes a query for uddiClient2 for relationships yet to be established; that is, looking for those relationships that the toKey side has not yet acknowledged.**

```
AssertionStatusReport assertionStatusReport2 =
    uddiClient2.getAssertionStatusReport(AssertionStatusItem.COMPLETION_STATUS_
    TOKEY_INCOMPLETE);
printOutXml("pending relationships for uddiClient2: case toKey incomplete",
            assertionStatusReport2);
```

- **Shows uddiClient2 agreeing to the peer-to-peer relationship requested by creating a publisherAssertion.**

```
System.out.println("");
System.out.println("uddiClient2 agrees to the peer-to-peer relationship ");
System.out.println("from bE1 to bE2");
dispositionReport = uddiClient2.addPublisherAssertion
    (createPeerToPeerPublisherAssertion(uddiClient2, bEKey1, bEKey2));
System.out.println("Done.");
```

- **Makes another query for uddiClient2 for relationships yet to be established to see if there are other peer-to-peer relationships to be established. There are no more pending relationships to be established.**

```
AssertionStatusReport assertionStatusReport2After =
    uddiClient2.getAssertionStatusReport
        (AssertionStatusItem.COMPLETION_STATUS_TOKEY_INCOMPLETE);
printOutXml("pending relationships for client2: toKey incomplete (should be
    none)", assertionStatusReport2After);
```

- **Finds related businesses that have established peer-to-peer relationships (that have published assertions) by calling the general inquiry message findRelatedBusinesses.**

```
RelatedBusinessesList rbList = uddiClient1.findRelatedBusinesses(bEKey1,
    createPeerToPeerKeyedReference(uddiClient1),
    null);
printOutXml("find all businesses that are peers to " + bEKey1, rbList);
```

- Deletes a publisher assertion relationship between bE1 and bE2, owned by uddiClient1.

```
System.out.println("");
System.out.println("Delete a publisherAssertion...");
dispositionReport = uddiClient1.deletePublisherAssertion
    (createIdentityPublisherAssertion(uddiClient1,bEKey1, bEKey2));
System.out.println("Done");
```

- Shows another way of deleting all publisher assertion relationships owned by uddiClient1 by using the setPublisherAssertions call.

```
System.out.println("");
System.out.println("Delete all publisherAssertions of uddiClient1 ");
System.out.println("by using setPublisherAssertions...");
publisherAssertions = uddiClient1.setPublisherAssertions(null);
printOutXml("Done. The current list:", publisherAssertions);
```

## OracleAS UDDI Registry Administration

The following sections describe OracleAS UDDI Registry administration features:

- [Using the Command-Line Tool uddiadmin.jar](#)
- [Configuring the Server](#)

### Using the Command-Line Tool uddiadmin.jar

As administrator, you perform many administrative operations using the command-line tool `uddiadmin.jar`.

The command-line tool `uddiadmin.jar` is located in the `${ORACLE_Home}/uddi/lib/uddiadmin.jar` file for UNIX and in the `%ORACLE_Home%\ORACLE%\uddi\lib\uddiadmin.jar` file for Windows. In general, the command-line tool uses the following command format:

```
java -jar uddiadmin.jar registry_admin_URL username password
[-verbose] options_and_their_parameters
```

In the format, the parameters have the following meanings:

- *registry\_admin\_URL*: A URL pointing to the administration end point:  
`http://OracleAS-host:OracleAS-port/uddi/admin`
- *username*: The default user name is `ias_admin`. The username must belong to the `uddiadmin` group.
- *password*: The default password is `ias_admin123`.
- `-verbose`: This causes stack trace information to be printed out when an exception is encountered.
- *options\_and\_their\_parameters*: Any of the options listed in [Table 10-3](#) and their parameters.

[Table 10-3](#) shows the command-line options for the command-line tool `uddiadmin.jar`.

**Table 10–3** *Command-Line Options for uddiadmin.jar*

Option	Description
<code>changeOwner</code>	Changes the ownership of the named entity to the specified user.
<code>correctChangeRecord</code>	Applies the <code>changeRecordCorrectionfile</code> file contents and <code>changeRecordNewDatafile</code> file contents to the UDDI node.
<code>deleteEntity</code>	Deletes the named entity irrespective of the owner of the entity.
<code>deleteRoleQuotaLimits</code>	Deletes the group-to-quota-limits mappings for the specified quota groups.
<code>destroyTModel</code>	Permanently deletes the named tModel from the registry.
<code>doPing</code>	Sends a UDDI replication <code>do_ping</code> message to the replication end-point URL specified.
<code>downloadReplicationConfiguration</code>	Downloads the currently used replication configuration from a specified UDDI node.
<code>getChangeRecord</code>	Gets the detail of a change record specified by <code>local_usn</code> .
<code>getHighWaterMarks</code>	Gets the high-water marks vector from the specified UDDI node.
<code>getProperties</code>	Lists the current registry configuration parameters.
<code>getRoleQuotaLimits</code>	Displays the current J2EE role-to-quota limit mappings.
<code>getUserDetail</code>	Retrieves the details of the named user.
<code>getUsers</code>	Lists all existing users who have entities in the registry.
<code>import</code>	Imports all <code>businessEntity</code> and <code>tModel</code> data structures and a <code>publisherAssertion</code> in the named file.
<code>setOperationalInfo</code>	Sets some operational information for entities.
<code>setProperty</code>	Changes the value of the named configuration parameter.
<code>setRoleQuotaLimits</code>	Sets the quota limit for the specified quota group.
<code>setWalletPassword</code>	Sets the wallet password to be used for HTTPS communication among UDDI nodes for UDDI replication.
<code>transferCustody</code>	Transfers the custody of a <code>tModel</code> or <code>businessEntity</code> data structure to a new operator and a new authorized name.
<code>uploadReplicationConfiguration</code>	Uploads the specified replication configuration to a particular UDDI node.

For reference information about these options, see "[Command-Line Options for the uddiadmin.jar Tool](#)" on page 10-62.

## Configuring the Server

You use the following options of the command-line tool `uddiadmin.jar` to configure the server:

- `getProperties`: Lists the current registry configuration parameters. The following shows an example:

```
java -jar uddiadmin.jar http://OracleAS-host:port/uddi/admin username password
[-verbose] -getProperties
```

See ["getProperties"](#) on page 10-64 for more information on this option.

- `setProperty`: Changes the value of the named configuration parameter. The OracleAS UDDI Registry J2EE application needs to be restarted for the parameters to take effect.

---



---

**Caution:** Be very careful when using the `setProperty` option to change server configuration property values. Making an incorrect property setting could cause severe damage to the integrity of the registry.

---



---

With the `setProperty` option, you can specify server configuration as described in ["Modifying Properties at Installation or First-Use"](#) on page 10-7.

## Managing Users

OracleAS UDDI Registry uses Oracle Internet Directory (OID) of the OracleAS Infrastructure as the default user repository. This is achieved through the use of LDAP-based provider of OC4J Java Authentication and Authorization Service (JAAS).

UDDI-specific OID groups are located under the `cn=uddi_groups` subtree of the group subtree of the OID default subscriber, and users are located under the user subtree of the OID default subscriber.

[Table 10-4](#) summarizes the groups of UDDI users.

**Table 10-4** *Default UDDI User Groups*

Group	Description
uddiadmin	Can access the administration end points and perform administrative activities.
uddipublisher	Can perform all activities specified in the <code>uddipublisher</code> group. Can access the publishing end point and save, update, or delete UDDI entities in the registry.
uddireplicator	Can perform replication activities based on the replication schedule: send replication requests such as <code>getChangeRecord</code> to other UDDI nodes and apply the <code>changeRecords</code> received.

---



---

**Note:** Do not remove any of these default UDDI groups.

---



---

In addition to the default UDDI groups, there are also a set of default groups for user quota purposes. As administrator, you can add, update, or remove the groups, based on the specific user quota policy that you may need to enforce.

By default, the users listed in [Table 10-5](#) are created during installation. You can add users to, or remove users from, these groups.



**Table 10–5** Default UDDI Users

Group	User Names	Comments
uddiadmin	ias_admin	Administration user. Typically, Oracle Enterprise Manager administrators also log in as <code>ias_admin</code> to publish to the UDDI registry through the Oracle Enterprise Manager integrated J2EE Web Services deployment and publishing wizard.
uddipublisher	uddi_publisher uddi_publisher1	Sample users for demonstrating publishing and different default quota groups.
uddireplicator	uddi_replicator	The default user that OracleAS UDDI Registry uses to perform UDDI replication activities in the background. This user should not be removed. If you do need to remove this user, make sure you add another user to the <code>uddireplicator</code> group. The user to start the Replication Client module must be updated as well, by modifying the <code>orion-application.xml</code> file in the <code>oraudrepl.ear</code> application.

Generic user management, such as creation, deletion, suspension, is handled by OID and its Oracle Delegated Administration Services. Refer to *Oracle Identity Management Guide to Delegated Administration* for more information.

User management, including operations such as creation, deletion, suspension, role management, is handled by the JAAS service of OC4J. Refer to *Oracle Application Server Containers for J2EE Services Guide* for more information.

However, to find out the authorized name of a user, use the `getUsers` or `getUserDetail` option of the `uddiadmin.jar` command-line tool:

- `getUsers`: Lists all existing users who have entities in the registry. For example:

```
java -jar uddiadmin.jar http://OracleAS-host:port/uddi/admin username password
[-verbose] -getUsers
```

See "[getUsers](#)" on page 10-65 for more information on this option.

- `getUserDetail`: Retrieves the details of the named user, currently the authorized name of each user. For example:

```
java -jar uddiadmin.jar http://OracleAS-host:port/uddi/admin username password
[-verbose] -getUserDetail username
```

See "[getUserDetail](#)" on page 10-64 for more information on this option.

## Enforcing Quotas

OracleAS UDDI Registry provides a mechanism to enforce the number of entities a publisher can own. A publisher can own at most a specific number of `tModel`, `publisherAssertion`, `businessEntity`, and `businessService` data structures for each `businessEntity`, and `bindingTemplate` data structures for each `businessService`, depending upon the quota group associated with the publisher. The quota group is guided by the user group to which the publisher is assigned.

OracleAS UDDI Registry uses a group-based mechanism for assigning quota limits to a publisher. When a new publisher is added, the OracleAS UDDI Registry administrator must associate the publisher with a quota group. [Table 10–6](#) shows the predefined quota groups and quota limits for each entity that a publisher can own.

**Table 10–6** *Predefined Quota Groups and Their Limits*

Quota Group	Quota Limits per Entity				
	businessEntity	businessService per businessEntity	bindingTemplates per businessService	tModel	publisherAssertion
Default	1	4	2	100	10
uddi_unlimited_quota_group	Unlimited	Unlimited	Unlimited	Unlimited	Unlimited
uddi_lowlimits_quota_group	2	2	1	3	3
<Implicit> UDDI_Administrators	Unlimited	Unlimited	Unlimited	Unlimited	Unlimited

The explicit Default quota group cannot be deleted. Users who are OracleAS UDDI Registry administrators are always assigned unlimited quota.

As OracleAS UDDI Registry administrator, you can also update a quota group, add a new quota group, delete a quota group, view the lists of quota groups and their quota limits, and associate a publisher with a quota group. The following sections describe each of these administrator tasks.

### Updating the Limits of a Quota Group

To update the limits of a quota group, use the `setRoleQuotaLimits` option of the command-line tool `uddiadmin.jar`.

Set the quota limit value for the specified quota group. This option can be used to create a new group-to-quota-limit mapping or to update an existing mapping. The parameters are defined as follows:

- `roleName`—name of the quota group to map to the specified limits
- `maxBE`—maximum number of `businessEntity` data structures allowed
- `maxBSperBE`—maximum number of `businessService` data structures per `businessEntity` allowed
- `maxBTperBS`—maximum number of `bindingTemplate` data structures per `businessEntity` allowed
- `maxTM`—maximum number of `tModel` data structures allowed
- `maxPA`—maximum number of `publisherAssertion` data structures allowed

The value `-1` means unlimited.

The following shows the format of the command:

```
java -jar uddiadmin.jar http://OracleAS-host:port/uddi/admin username password
-setRoleQuotaLimits roleName maxBE maxBSperBE maxBTperBS maxTM maxPA
```

See "[setRoleQuotaLimits](#)" on page 10-67 for more information on this option.

### Adding a New Quota Group (Advanced Operation)

To add a new quota group, perform the following steps:

1. Oracle recommends that you back up the configuration files, `application.xml`, `web.xml`, and `orion-application.xml`, before you begin this process.
2. Add the group to the user store, typically OID.
3. Define the corresponding J2EE security role, `partnerGroup`, for the new group name you want to create in the `orauddi.ear` application. The settings must be

added in both the `application.xml` and `web.xml` files of the `orauddi.ear` application.

4. Define the J2EE security role to the user store mapping in the `orion-application.xml` file of the `orauddi.ear` application.
5. Define the actual limits of the quota group using the `setRoleQuotaLimits` option of the command-line tool `uddiadmin.jar`. See ["Updating the Limits of a Quota Group"](#) on page 10-38 for more information.

### Deleting a Quota Group (Advanced Operation)

To remove a quota group, perform the following steps:

1. Remove the J2EE security role for the `partnerGroup` you want to remove from the `orauddi.ear` application. The settings must be removed from both the `application.xml` and `web.xml` files of the `orauddi.ear` application.
2. Remove the J2EE security role to the user store mapping in the `orion-application.xml` file of the `orauddi.ear` application.
3. Remove the actual limits of the quota group using the `deleteRoleQuotaLimits` option of the command-line tool `uddiadmin.jar`, as shown in the following example:

```
java -jar uddiadmin.jar http://OracleAS-host:port/uddi/admin username password
    -deleteRoleQuotaLimits roleName [roleName...]
```

See ["deleteRoleQuotaLimits"](#) on page 10-62 for more information on this option.

4. Remove the group from the user store, typically OID.

### Viewing the Lists of Quota Groups and Their Limits

To view the list of quota groups and their limits, use the `getRoleQuotaLimits` option of the command-line tool `uddiadmin.jar`. This option displays all the J2EE-role-to-quota-limit mapping currently set in the registry, as shown in the following example:

```
java -jar uddiadmin.jar http://OracleAS-host:port/uddi/admin username password
    -getRoleQuotaLimits
```

See ["getRoleQuotaLimits"](#) on page 10-64 for more information on this option.

### Associating a Publisher with a Quota Group

When you add a user to the user store (OID or `jazn-data.xml`), you must place the user in a group so that it is assigned to the appropriate quota group. For example, with the pre-defined settings, administrators can assign a user to have the low quota limits by assigning the user to the `uddi_lowlimits_quota_group` group.

If a user does not belong to a particular group, the user is assigned the quota limits from the Default group. An OracleAS UDDI Registry administrator is always assigned unlimited quota.

## Managing Administrative Entities

Use the following options of the command-line tool `uddiadmin.jar` for administrative entity management:

- **deleteEntity**: Deletes the named entity irrespective of the owner of the entity. Note that this operation performs a nonpermanent delete (hide) operation in the case of a tModel entity.  
See "**deleteEntity**" on page 10-62 for reference information about this option.
- **destroyTModel**: Permanently deletes the named tModel from the registry (as opposed to the UDDI-defined delete\_tModel call, which is just hiding the tModel entity).  
See "**destroyTModel**" on page 10-63 for reference information about this option.
- **changeOwner**: Changes the ownership of the named entity to the new specified user.  
See "**changeOwner**" on page 10-62 for reference information about this option.

## Importing Entities

To import entities from a file, you use the `import` option of the command-line tool `uddiadmin.jar`. You can import all `businessEntity`, `tModel`, and `publisherAssertion` data structures in the named file.

To import the `businessEntity` data structure, the named file for importing should contain a UDDI `businessDetail` XML document.

To import `tModel` data structures, the named file should contain a UDDI `tModelDetail` XML document. By importing them, entity keys (such as `businessKey`, `serviceKey`, `bindingKey`, `tModelKey`) are preserved. The `operatorName` and `authorizedName` fields, however, are not preserved. The `operatorName` field will be replaced by the `operatorName` configuration parameter of the registry. The owner of the imported entities is the administrator; hence, the `authorizedName` field will be the `authorizedName` of the administrator.

The `import` option is particularly useful in importing the well-known service interface specification `tModel` and classification `tModel` data structures from some authoritative sources.

Because the entity keys are preserved, you should be careful in evaluating the source of the entities to ensure there will not be a collision in entity keys.

For importing a `publisherAssertion`, two Boolean values are required. These Boolean values are used to indicate from which side (or both sides, when two Boolean values are true) the `publisherAssertion` is going to be inserted.

You can import in single mode, using the option `-s`, which does not allow partial success (some entities are imported and some are not, due to some error condition), or in multiple mode (`-m`), which does allow partial success.

The following shows the format of the `import` option:

```
-import [-s|-m] {-businesses filename | -tmodels filename |
  -assertions filename -fromBusinessCheck {true|false}
  -toBusinessCheck {true|false} }
```

For example, the contents of the `publisherAssertion` file, `assert.xml`, could contain the following:

```
<publisherAssertion generic="2.0" xmlns="urn:uddi-org:api_v2">
  <fromKey>22A5A0304C64-11D8-AB19-B8A03C50A862</fromKey>
  <toKey>27CC6702-7F6E-4395-A0B8-97D2FFB5F7634 </toKey>
  <keyedReference tModelKey="UUID:807A2C6A-EE22-470D-ADC7-E0424A337C03"
    keyName="subsidiary">
```

```

        keyValue="parent-child" />
</publisherAssertion>

```

Then, to import the publisherAssertion data structure, use the following command:

```

java -jar uddiadmin.jar http://OracleAS-host:port/uddi/admin username password
-import -s -assertions assert.xml -fromBusinessCheck true
-toBusinessCheck true

```

## Setting Operational Information

You can use the `setOperationalInfo` option to set some operational information of entities, such as the operator name, authorized name, or timestamp of a businessEntity or tModel specified by a key, for example, following an import operation.

The `setOperationalInfo` option uses two syntax formats:

- To change the operator name, the authorized name, or the timestamp, or all three, of a businessEntity or tModel specified by a key, use the following format:

```

-setOperationalInfo {-businessKey key | -tModelKey key}
[-newOperator OperatorName] [-newAuthorizedname authName] [-newTime timestamp]

```

Any combination of operator name, authorized name, or timestamp of a businessEntity or tModel data structure is allowed.

- To change only the timestamp of a businessService or bindingTemplate data structure, use the following format:

```

-setOperationalInfo {-serviceKey key | -bindingKey key} [-newTime timestamp]

```

See "[setOperationalInfo](#)" on page 10-66 for more information on this option.

## UDDI Replication

OracleAS UDDI Registry allows administrators to create a logical registry that comprises one or more OracleAS UDDI Registry implementations, as well as UDDI implementations from other vendors that also implement the UDDI v2 Replication Specification.

This section briefly describes the data replication process and the program interface required to achieve complete data replication among UDDI operator nodes that form a UDDI service. UDDI replication ensures that all operator nodes see all the changes that have originated at individual operator nodes. In addition, any inquiries made at any operator node within the UDDI service yield results consistent to those made at any other operator node within the UDDI service, hence the logical OracleAS UDDI Registry.

For detailed technical descriptions of concepts and definitions involved with UDDI replication, including replication processing, how to bring new UDDI operators online, checking and validation of replicated data, see the UDDI v2.0.3 Replication Specification. The sections that follow describe the Oracle implementation of UDDI replication.

### Enabling UDDI Replication

To enable UDDI replication, as administrator, you must perform the following steps:

1. Participate with and agree to the replication topology with UDDI administrators of other operator nodes. This involves editing the replication configuration (in the format specified in the UDDI v2 Replication Specification), and using the

uploadReplicationConfiguration and downloadReplicationConfiguration options of the command-line tool uddiadmin.jar.

```
java -jar uddiadmin.jar http://OracleAS-host:port/uddi/admin username password
-uploadReplicationConfiguration xml_file_containing_replication_configuration
```

See "[uploadReplicationConfiguration](#)" on page 10-68 for reference information on this option.

Before you can download successfully, you must upload the replication configuration.

```
java -jar uddiadmin.jar http://OracleAS-host:port/uddi/admin username password
-downloadReplicationConfiguration
```

See "[downloadReplicationConfiguration](#)" on page 10-63 for reference information on this option.

2. Enable replication scheduling by setting the property status to the value 1:

```
java -jar uddiadmin.jar http://OracleAS-host:port/uddi/admin username password
-setProperty oracle.uddi.server.scheduler.status=1
```

See "[setProperty](#)" on page 10-66 for reference information on this option. See "[status](#)" on page 10-79 for reference information on this property.

3. Enable update journal storage by setting the property startMaintainingUpdateJournal, to true:

```
java -jar uddiadmin.jar http://OracleAS-host:port/uddi/admin username password
-setProperty oracle.uddi.server.replication.startMaintainingUpdateJournal=true
```

See "[setProperty](#)" on page 10-66 for reference information on this option. See "[startMaintainingUpdateJournal](#)" on page 10-78 for reference information on this property.

After UDDI replication is started, as administrator, you can suspend or resume replication operations by stopping or starting the `oraudrepl.ear` application.

If HTTPS client certification is used, you must do the following:

1. Obtain an exported Oracle wallet file using Oracle Wallet Manager and specify the exported wallet location by setting the property `walletLocation`. In the following example, the location of `ewallet.p12` is relative to `${ORACLE_Home}/uddi/config` on UNIX or `%ORACLE_Home_ORACLE%\uddi\config` on Windows:

```
java -jar uddiadmin.jar http://OracleAS-host:port/uddi/admin username password
-setProperty oracle.uddi.server.replication.walletLocation=ewallet.p12
```

This option needs to be set only once.

See "[setProperty](#)" on page 10-66 for reference information on this option. See "[walletLocation](#)" on page 10-81 for reference information on this property.

2. Use the `setWalletPassword` option to supply the wallet password, whenever the `oraudrepl.ear` application is started or restarted. Specify the `uddirepl/admin/wallet` path, as shown in the following example:

```
java -jar uddiadmin.jar http://OracleAS-host:port/uddirepl/admin/wallet
username password
-setWalletPassword=walletpassword
```

Because the password is not persistent for security reasons, each time the application is restarted, this option must be invoked.

See "[setWalletPassword](#)" on page 10-67 for reference information on this option.

In some cases, the administrator of the source of the error must correct an invalid `changeRecord` operation that caused the error. The administrator can use the `correctChangeRecord` option of the command-line tool `uddiadmin.jar` to supply the correct `changeRecord` data. See "[Handling Replication Exceptions](#)" on page 10-43 for more information.

### Transferring Custody

To transfer the custody of a `tModel` or a `businessEntity` to a new operator and a new authorized name use the `transferCustody` option of the command-line tool `uddiadmin.jar`. This option is part of custody transfer as defined by the UDDI specification. See "[transferCustody](#)" on page 10-68 for reference information on this option.

### Setting Properties for the UDDI Replication Scheduler

You can use the following UDDI server properties to set UDDI replication scheduler properties:

- **timer\_pool\_size**: Specifies the number of concurrently active threads used by the scheduler. The following example sets the number of threads to 1:

```
java -jar uddiadmin.jar http://OracleAS-host:port/uddi/admin username password
-setProperty oracle.uddi.server.scheduler.timer_pool_size=1
```

See "[timer\\_pool\\_size](#)" on page 10-80 for reference information on this property.

- **status**: Indicates whether or not the scheduler is enabled to send out replication requests. The value 0 sets the scheduler off; the value 1 sets it to on. The following example sets it to on:

```
java -jar uddiadmin.jar http://OracleAS-host:port/uddi/admin username password
-setProperty oracle.uddi.server.scheduler.status=1
```

See "[status](#)" on page 10-79 for reference information on this property.

### Handling Replication Exceptions

If any errors occur during replication operations, OracleAS UDDI Registry logs the error in the `application.log` file of the `oraudrepl.ear` application. You should investigate the cause of the error and correct each problem.

To correct the change records, use the `correctChangeRecord` option. This option applies the `changeRecordCorrectionfile` file contents and `changeRecordNewDatafile` file contents to the UDDI node. The content of these files must conform to the UDDI replication XML schema. This option is part of UDDI replication error recovery.

### Advanced Configuration and Tuning for UDDI Replication

In the addition to the UDDI server properties described in previous sections, you can use the following server properties with replication:

- **changeRecordWantsAck**: Controls whether or not ACK is required for the change records sent out from the local node. See "[changeRecordWantsAck](#)" on page 10-71 for reference information on this property.

- **maxChangeRecordsSentEachTime**: Controls the maximum number of change records sent out in response to an incoming `getChangeRecords` request. See "[maxChangeRecordsSentEachTime](#)" on page 10-74 for reference information on this property.
- **pushEnabled**: Controls whether or not a push task should be performed for UDDI replication. See "[pushEnabled](#)" on page 10-76 for reference information on this property.
- **pushTaskExecutionPeriod**: Controls the push task execution period (in milliseconds). See "[pushTaskExecutionPeriod](#)" on page 10-76 for reference information on this property.
- **soapRequestAuthMethod**: Controls the authentication method the registry node will try to use in sending replication SOAP requests to other nodes. If HTTP client certification (`CLIENT_CERT`) is used, you must set the wallet password each time the registry node gets started or restarted.  
See "[soapRequestAuthMethod](#)" on page 10-77 for reference information on this property.
- **soapRequestTimeout**: Controls the timeout value for each SOAP replication request (in milliseconds). See "[soapRequestTimeout](#)" on page 10-78 for reference information on this property.
- **taskExecutionPeriod**: Controls the period of time during which replication task should be executed (in milliseconds). See "[taskExecutionPeriod](#)" on page 10-80 for reference information on this property.

You can use the following options of the command-line tool `uddiadmin.jar` to perform advanced configuration:

- **doPing**: Sends a UDDI replication `do_ping` message to the replication end-point URL specified. This is similar to the ping command in TCP/IP that is used to check if the other end point is active. The optional `walletPassword` parameter is useful when the JVM, which receives the `do_ping` message, does not have a valid wallet password set.  
See "[doPing](#)" on page 10-63 for reference information on this option.
- **getChangeRecord**: Gets the detail of a change record specified by `local_usn` (an integer). This API is used in conjunction with the `correctChangeRecord` option to correct wrong or inconsistent data across different UDDI nodes with OracleAS UDDI Registry.  
See "[getChangeRecord](#)" on page 10-63 for reference information on this option.
- **getHighWaterMarks**: Gets the high-water marks vector from the specified UDDI node. The optional `walletPassword` parameter is useful when the JVM, which receives the `do_ping` message, does not have a valid wallet password set.  
See "[getHighWaterMarks](#)" on page 10-64 for reference information on this option.

## Registry-Based Category Validation

OracleAS UDDI Registry can perform a spell-check form of category value validation. As administrator, you can add or remove the set of categories that will be validated by the registry. Refer to the UDDI v2 specification for more information.



## Adding a New Category for Registry-Based Validation

To add a new category, you must load the category values into the database and register the category with the registry. Perform the following steps:

1. Publish the category to the registry by saving a new tModel data structure. For example, look at the tModel data structure named `ntis-gov:naics:1997`. You can use a third-party tool or the included sample Web applications link:

```
http://OracleAS-host:OracleAS-port/uddi/
```

If the tModel data structure has been defined in some other registry, you can also import it (instead of creating a new one, which results in different tModelKey entities) using the `uddiadmin.jar` command-line tool. See ["Importing Entities"](#) on page 10-40 for more information on the import operation.

The tModel data structure published should be classified as *"unvalidatable"* in the `uddi-org:types` taxonomy. Specifically, the following `keyedReference` should appear in the `CategoryBag` element of the tModel data structure:

```
<keyedReference tModelKey="uuid:C1ACF26D-9672-4404-9D70-39B756E62AB4"
keyName="" keyValue="unvalidatable" />
```

2. Load the category values into the database. To do this, all the category values should be in a file using the following format:
  - Each line of the file describes one category value in the category. It should be in the following format:

```
<category value> | <description of category value>
| <category value of the parent>
```

- If a category value is a root value, for example, it has no parent, the category value of the parent should be set to itself.
- The line in the file for a category value should occur before the lines for all of its descendants.

Examples can be found in the `uddi/taxonomy` directory for UNIX and in the `uddi\taxonomy` directory for Windows. Excerpts from the NAICS file are as follows:

```
22|Utilities|22
221|Utilities|22
2211|Electric Power Generation, Transmission|221
```

If your files use different characters from different languages, it is recommended that you save the file with UTF-8 encoding to avoid any problems that may arise, such as character corruption.

3. Create a SQL\*Loader control file to load the category file. An example is `${ORACLE_Home}/uddi/admin/naics-97.ctl` for UNIX and `%ORACLE_Home_ORACLE%\uddi\admin\naics-97.ctl` for Windows. Copy the file and replace the category file name in the control file with the name of the one you create. Refer to the UDDI v2 specification for more information about generating a unique ID for the new category tModel.
4. Load the category file into the database using SQL\*Loader. Refer to *Oracle Database Utilities*, part of the Oracle Database documentation, for more information about using SQL\*Loader.
5. Configure the registry so that it recognizes the category that must be validated by using the command-line tool, `uddiadmin.jar`. For example, to add a new

tModel entity with key UUID: FFFFFFFF-FFFF-FFFF-FFFF-FFFFFFFF0, use the `setProperty` option to set the property `categoryValidationTModelKeys` as follows:

```
java -jar uddiadmin.jar http://OracleAS-host:port/uddi/admin username password
-setProperty "oracle.uddi.server.categoryValidationTModelKeys=
'UUID:C1ACF26D-9672-4404-9D70-39B756E62AB4',
'UUID:4E49A8D6-D5A2-4FC2-93A0-0411D8D19E88',
'UUID:C0B9FE13-179F-413D-8A5B-5004DB8E5BB2',
'UUID:CD153257-086A-4237-B336-6BDCBDC6634',
'UUID:FFFFFFF-FFFF-FFFF-FFFF-FFFFFFFF0' "
```

Make sure that you enter the command on one line, with no returns or extra spaces.

Because the `setProperty` option defines all categories that need to be validated, you must specify all the existing `tModelKey` values plus the new `tModelKey` value to add a new category.

You can also use the following server properties:

- **identifierValidation**: Controls validation for all `IdentifierBag` entities.
- **operatorCategory**: Determines whether or not additional entities may be categorized as an operator node, if the value of `categoryValidation` is true.
- **categoryValidation**: Controls validation for all `CategoryBag` entities.
- **assertionKeyedRefValidation**: Controls validation for all `publisherAssertion` `keyedReference` data structures.
- **tModelInstanceInfoKeyValidation**: Determines if `tModelKey` existence validation occurs within `tModelInstanceInfo` elements.
- **addressTModelKeyValidation**: Determines if `tModelKey` existence validation occurs within `address` elements.
- **hostingRedirectorValidation**: Determines if `hostingRedirector` validation occurs within `bindingTemplate` elements. Validation ensures that the referenced `bindingTemplate` element exists and does not contain a `hostingRedirector` element.

See "[Server Configuration Properties](#)" on page 10-69 for information on these properties.

6. Allow the registry users to use the category `tModel` published by removing the "*unvalidatable*" categorization done in Step 1. Specifically, the following `keyedReference` element should be removed from the `CategoryBag` element of the `tModel` data structure:

```
<keyedReference tModelKey="uuid:C1ACF26D-9672-4404-9D70-39B756E62AB4"
keyName="" keyValue="unvalidatable" />
```

### Removing a Category from Registry-Based Validation

To remove a category from registry-based validation, you should unregister the category and remove the category values from the database. Perform the following steps:

1. To unregister the category from the registry, remove it from the list of validated categories using the `uddiadmin.jar` command with the `setProperty` option to set the property `categoryValidationTModelKeys`.

You do not have to (and in general should not) delete the tModel data structure from the registry.

2. To remove the category values from the database, use the SQL\*Plus script `wurvcrm.sql` in the `uddi/admin` directory for UNIX and in the `uddi\admin` directory for Windows. For example:

```
sqlplus sys/sys-password @wurvcrm.sql
```

When running this script, you will be prompted for the tModelKey value of the category to be removed. You should see that a set of rows has been deleted. If the result shows that 0 rows were deleted, you entered an invalid tModelKey value. Run the script again.

## External Validation

Third parties can register new category and identifier schemes, and then control the validation process used by OracleAS UDDI Registry to perform external validation or checking. This enables a third-party category provider to validate the UDDI entities to be saved when the entity is categorized, or identified with the category, by providing a `validate_values` SOAP Web service.

The operator that calls the `validate_values` service passes a `businessEntity`, a `businessService`, or a tModel element as the sole argument to this call. This is the same data that is being passed within a `save_business`, `save_service`, or `save_tModel` API call. External validation is performed for any third-party category provider and identifier scheme that is classified as checked. A tModel element marked as checked asserts that it represents a categorization, identifier, or namespace tModel element that has a properly registered validation service.

If no error is found, the response is a `dispositionReport` message returning an `errorCode` value of `E_success` and an `errno` value of 0. If any error is found, or the called service needs to signal that the information being saved is not valid based on the validation algorithm chosen by the external service provider, then the service should raise a SOAP Fault and indicate either an `errorCode` value of `E_invalidValue` or `E_valueNotAllowed`. In either case, the error text indicates the `keyedReference` data that is being rejected, and the reason why.

Use the command-line tool `uddiadmin.jar` with the `setProperty` option to:

- Enable external validation
- Add an externally validated category to the registry
- Remove an externally validated category from the registry

### Enabling External Category Validation

To enable external category validation, issue the `setProperty` option to set the server property `externalValidation` as follows:

```
java -jar uddiadmin.jar http://OracleAS-host:port/uddi/admin username password
-setProperty oracle.uddi.server.externalValidation=true
```

### Adding an Externally Validated Category to the Registry

To add an externally validated category to the registry, perform the following steps:

1. Publish the new category as a tModel data structure to the registry. This data structure must be classified as checked under the `uddi-org:types` category.

2. Register the external validation service of the category with the registry by updating the server property `externalValidationTModelList` using the `setProperty` option as follows:

```
java -jar uddiadmin.jar http://OracleAS-host:port/uddi/admin username password
-setProperty oracle.uddi.server.externalValidationTModelList=key-value,
URL-validation-service
```

For example, if the category tModel published has the key "uuid:acme-taxonomy-key", and the URL of the validation service is `http://acme.com/externalValidation`, use the following command:

```
java -jar uddiadmin.jar http://OracleAS-host:port/uddi/admin username password
-setProperty
oracle.uddi.server.externalValidationTModelList=uuid:acme-taxonomy-key,
http://acme.com/externalValidation
```

3. Optionally, you can tune the timeout limit (in milliseconds) for calls to the external validation service using the server property `externalValidationTimeout` as follows:

```
java -jar uddiadmin.jar http://OracleAS-host:port/uddi/admin username password
-setProperty oracle.uddi.server.externalValidationTimeout=5000
```

### Removing an Externally Validated Category from the Registry

To remove an externally validated category from the registry, perform the following steps:

1. Update the server property `externalValidationTModelList` using the `setProperty` option by supplying a null value for the `URL-validation-service` as follows:

```
java -jar uddiadmin.jar http://OracleAS-host:port/uddi/admin username password
-setProperty oracle.uddi.server.externalValidationTModelList=key-value, ""
```

For example, if the category tModel published has the key "uuid:acme-taxonomy-key", and the URL of the validation service is `http://acme.com/externalValidation`, the command with the null entry will be as follows:

```
java -jar uddiadmin.jar http://OracleAS-host:port/uddi/admin username password
-setProperty
oracle.uddi.server.externalValidationTModelList=uuid:acme-taxonomy-key, ""
```

2. Deprecate or update the corresponding tModel data structure. If the tModel is not updated, the registry will reject any new UDDI entries that are categorized or identified by the category that was removed in subsequent `save` calls to the `save_business`, `save_service`, or `save_tModel` API.

## Performance Monitoring and Tuning

On the back end of an Oracle database, UDDI servlets and the associated JDBC connection pools can be monitored using Oracle Enterprise Manager 10g and other standard database monitoring and tuning utilities.

In an OC4J standalone environment, performance information is typically available at:

```
http://oc4j-host-name:port-number/dmsoc4j/Spy
```

## Data Backup and Restore Operations

You can back up and restore UDDI Registry data by using the standard Oracle database back up and restore operations. See the *Oracle Backup and Recovery Concepts*, part of the Oracle Database documentation set.

## Database Configuration

The following sections describe some database-specific configuration information.

### Database Character Set Should Be UTF-8

The database character set should be UTF-8 to accommodate all possible characters. However, you are absolutely certain that the data to be stored in the registry contains characters of a specific country or region (such as western Europe), you can use the appropriate database character set.

### Database Character Set and Built-in ISO-3166 Classification

The UDDI specification mandates that the registry support the full UTF-8 character set. Oracle recommends, though does not require, using UTF-8 as the character set for the Oracle Application Server Infrastructure database if OracleAS UDDI Registry is used.

If the database is not configured with the UTF-8 character set or its equivalent or superset, there could be data corruption and error due to loss in character set conversion to or from UTF-8. Refer to the *Oracle Globalization Support Guide* for details.

In particular, the descriptions in the UDDI built-in ISO-3166 classification contain descriptions with non-ASCII characters, such as some Western European characters and some Eastern European characters for the names of cities or regions. In order to support the non-UTF-8 database, all non-ASCII characters in the descriptions are replaced with ASCII characters as an approximation.

If you do have a UTF-8 database, you can upgrade the built-in ISO-3166 classification to the one with accurate descriptions using the following instructions:

- Delete the existing ISO-3166 classification by running the SQL script, `wurrmiso.sql`, for example:

```
cd ${ORACLE_Home}/uddi/admin
sqlplus system/manager @wurrmiso.sql
```

- Load the ISO-3166 classification with accurate descriptions by using SQL\* Loader control file `iso3166-99ctl`, for example:

```
cd ${ORACLE_Home}/uddi/admin
sqlldr userid=system/manager control=iso3166-99ctl
```

### Functional Index Must Be Enabled

The functional index must be enabled to support index-based, case-insensitive search. The following initialization parameter is involved: `query_rewrite_enabled=true`

In addition, the cost-based optimizer must be turned on for analyzing all tables or indexes in the UDDISYS schema. For example:

```
execute dbms_stats.gather_schema_stats(ownname=>'UDDISYS',cascade=>true);
```

### Accuracy of Modified Timestamps of UDDI Entities

The accuracy of modified timestamps of UDDI entities is dependent on the version and compatibility of the database. If the database compatibility is release 9.0.1 or later, the modified timestamps are of SQL type `TIMESTAMP`, with accuracy up to microseconds. If the database compatibility is prior to release 9.0.1, the modified timestamps are of SQL type `DATE`, with accuracy up to seconds.

## Transport Security

In general, the Inquiry API does not require authentication. However, if the inquiry end point needs to be protected, transport-level authentication, such as HTTP BASIC authentication and HTTPS Secure Sockets Layer (SSL) client authentication, can be enabled by configuring the `web.xml` file. A security role, `uddiguest`, is reserved for accessing the protected inquiry end point. Refer to *Oracle Application Server Containers for J2EE Services Guide* and *Oracle Application Server Containers for J2EE User's Guide* for more information about security roles and related security configuration.

For the publishing end-point URL, consider allowing only HTTPS access. To disable HTTP access, edit the `web.xml` file of the `orauddi` application to enforce data confidentiality and make adjustments to HTTP servers. For example, to disable HTTP access in the `web.xml` file, use the following code:

```
<user-data-constraint>
  <transport-guarantee>CONFIDENTIAL</transport-guarantee>
</user-data-constraint>
```

Refer to the chapter on security in *Oracle Application Server Containers for J2EE User's Guide* and to *Oracle Application Server Containers for J2EE Services Guide* for more information.

Similarly, you can set up HTTPS access for the administrative end point and the UDDI Replication end point in the same way.

## UDDI Open Database Support

In addition to the Oracle Application Server Infrastructure database, the following databases are supported by OracleAS UDDI Registry:

- [Microsoft SQL Server](#)
- [IBM DB2](#)
- [Other Oracle Database \(Non-Infrastructure\)](#)

For Microsoft SQL Server and IBM DB2, the Oracle Application Server DataDirect Connect JDBC driver is needed.

The steps described in the following sections assume that the relevant database server has been installed. These instructions also assume that Oracle Application Server Portal has been installed, which copies the relevant UDDI files to `${ORACLE_Home}/uddi/admin` on UNIX or `%ORACLE_Home%\ORACLE%\uddi\admin` on Windows.

---

**Note:** You can also deploy OracleAS UDDI Registry in a standalone OC4J installation. See the *Oracle UDDI Support for Web Services Readme for Standalone Kit* on OTN for more information:

<http://www.oracle.com/technology/tech/webservices/htdocs/uddi/index.html>

As you follow the instructions in that document, you can specify arbitrary values for the `db.host`, `db.port`, and `db.sid` options to the `ant` command. Then, you will change those values later when you define the data source in the following sections.

---

## Microsoft SQL Server

The following sections describe installation and configuration information with SQL Server.

### Script Source Directory

Installation must be performed from a Windows machine. If the `%ORACLE_Home%\uddi\admin\mssql` directory is not accessible from the SQL Server machine, then copy this directory to a location that is accessible. This directory (or the original `%ORACLE_Home%\uddi\admin\mssql` if no copying is necessary) will be referred to as `%MSSQL_HOME_DB%`.

### Create the Database and User

The `%MSSQL_HOME_DB%\wurcreatedb_mssql.sql` script has been provided to create the `uddisys` database and `uddisys` user for a SQL Server instance in mixed-authentication mode. If you are using Windows authentication or wish to alter some of the settings in this script, you may do so as long as all the following requirements are met:

- The collation for the `uddisys` database must be case-sensitive.
- Recursive triggers must be enabled on the `uddisys` database.
- The `uddisys` user must have the `uddisys` database as its default database.
- The `uddisys` user must be a member of the `db_owner` role for the `uddisys` database.

To run the script with the Microsoft `osql` utility, use the administrator login (`sa`) and the administrator password. The following example assumes that the administrator password is `sa`, but if it is not, please substitute the appropriate password for your environment:

```
osql -S server -U sa -P sa -i wurcreatedb_mssql.sql
```

In the example, `server` is the server hosting the SQL Server instance.

If you receive the following error, make sure to change the authentication mode for SQL Server to SQL Server and Windows mode:

```
"Login failed for user 'sa'. Reason: Not associated with a trusted SQL Server Server Connection."
```

To change the authentication mode, open the SQL Server enterprise manager, navigate to your server, right-click and choose properties, click the security tab and select SQL Server and Windows. Then, restart SQL Server.

## Install the Schema

Go to the `%MSSQL_HOME_DB%` directory. Use the `osql` utility to execute the SQL script `wurinst_mssql.sql` using the `uddisys/uddisys` account created in "Create the Database and User" on page 10-51.

The syntax is as follows:

```
osql -S server -U user -P password -d database -i wurinst_mssql.sql
```

In the example, `server` is the server hosting the SQL Server instance.

For example:

```
osql -S server-machine -U uddisys -P uddisys -d uddisys -i wurinst_mssql.sql
```

## Import BUILTIN\_CHECKED\_CATEGORY Table Entries

Import the `iso3166-99_tModelKey.txt`, `naics-97_tModelKey.txt`, and `unspsc-73_tModelKey.txt` files from the `%MSSQL_HOME_DB%` directory into the `BUILTIN_CHECKED_CATEGORY` table as follows:

1. Select the **Import and Export Data** option from the SQL Server Start menu options. Click **Next**.
2. For the Data Source, select the last option, **Text File**. Then, provide the name and location of the appropriate text file, `%MSSQL_HOME_DB%\iso3166-99_tModelKey.txt`. Click **Next**.
3. The default file format should be **Delimited**. Accept this by clicking **Next**.
4. Set the delimiter to the vertical bar character (`|`). Click **Next**.
5. Select the **uddisys** database for the destination. Provide the appropriate authentication mechanism and credentials, which are SQL Server Authentication with user `uddisys` and password `uddisys`, by default. Make sure that the selected database is **uddisys**. Click **Next**.
6. Click the **Destination** and select the `BUILTIN_CHECKED_CATEGORY` table.
7. Click **Transform**. Map `TMODEL_KEY` to `Col001`, `KEY_NAME` to `Col003`, `KEY_VALUE` to `Col002`, and `PARENT_VALUE` to `Col004`. Click **OK**.
8. Click **Next**.
9. Click **Next** to run immediately, and click **Finish** to start.
10. Repeat this process for the `naics-97_tModelKey.txt` and `unspsc-73_tModelKey.txt` files.

---

**Note:** If the character set of your database is not UTF-8, do not use the script `iso3166-99.txt` to load the ISO-3166 taxonomy because the taxonomy contains characters from different languages. Instead, use the script `iso3166-99-ascii.txt` to load an ASCII-only version of the taxonomy.

---

## Configure OC4J to Use SQL Server

Define a data source with the name and location set to `jdbc/OracleUddi` to reflect that SQL Server is the desired database, similar to the following:

```
<data-source
  class="com.evermind.sql.DriverManagerDataSource"
  name="jdbc/OracleUddi"
```



```

location="jdbc/OracleUddi"
connection-driver="com.oracle.ias.jdbc.sqlserver.SQLServerDriver"
username="uddisys"
password="uddisys"
url="jdbc:oracle:sqlserver://server:1433;SelectMethod=cursor;User=uddisys;Password=uddisys"
/>

```

Note that *server* is the network name or IP address of the server hosting the SQL Server instance used for OracleAS UDDI Registry. Be sure you enter the line that begins `url=` on *one* line.

The data source needs to be accessible by the `orauddi.ear` and `oraudrepl.ear` applications.

Refer to the Data Sources chapter in the *Oracle Application Server Containers for J2EE Services Guide* for more information.

Restart the UDDI server for these changes to take effect.

## IBM DB2

The following sections describe installation and configuration information for OracleAS UDDI Registry relative to IBM DB2.

### Script Source Directory

If the `${ORACLE_Home}/uddi/admin/db2` directory is not accessible from the machine with the relevant DB2 tools, then copy this directory to a location that is accessible. This directory will be referred to as `${DB2_HOME_DB}` on UNIX or `%DB2_HOME_DB%` on Windows.

### Create the Database and User

Go to the `${DB2_HOME_DB}` directory on UNIX or the `%DB2_HOME_DB%` directory on Windows. The `wurcreatedb_db2.sql` script is provided for creating the `uddisys` database. The user is responsible for creating a `uddisys` user with password `uddisys` based on the authentication scheme that is being used for DB2. By default, this requires creating a `uddisys` user at the operating system level. On Windows, the `uddisys` user should belong to the local administrator group.

If you wish to alter some of the settings in this script, you may do so as long as both the following requirements are met:

- The default tablespace for the `uddisys` database must be at least 8 KB pages. This also requires providing a buffer pool that will support a page size of at least 8 KB.
- The value of the `applheapsz` parameter must be increased to approximately 12800 pages.

To run the script, start the DB2 Command Line Processor by entering `db2` in UNIX or `db2cmd` in Windows. Then, execute the script:

```
db2 -t +p < wurcreatedb_db2.sql
```

The option `-t` allows the use of semicolons to terminate SQL statements and `+p` suppresses prompting.

### Install the Schema

Run the `wurinst_db2.sql` script. This also triggers the `wurcreat.sql`, `wurdbsql.sql`, and `wurpopul.sql` scripts.

To run these scripts, launch the command-line processor as previously described, then enter the following:

```
db2 -t +p < wurinst_db2.sql
```

### Import BUILTIN\_CHECKED\_CATEGORY Table Entries

Import the `iso3166-99_tModelKey.txt`, `naics-97_tModelKey.txt`, and `unspsc-73_tModelKey.txt` files into the `BUILTIN_CHECKED_CATEGORY` table as follows:

1. Right-click the table `BUILTIN_CHECKED_CATEGORY` from the Control Center and select **IMPORT**.
2. Specify the Import file as `${DB2_HOME_DB}/iso3166-99_tModelKey.txt` for UNIX or `%DB2_HOME_DB%\iso3166-99_tModelKey.txt` for Windows.
3. Select **Delimited ASCII format (DEL)**. Click **Options** and select the vertical bar character (|) as the delimiter for Column delimiter (COLDEL).
4. Use the INSERT import mode (the default).
5. Set the Commit records equal to 500.
6. For the Message file, enter `${DB2_HOME_DB}/uddi/admin/db2/iso3166-99_tModelKey.log` for UNIX or `%DB2_HOME_DB%\uddi\admin\db2\iso3166-99_tModelKey.log` for Windows.
7. Go to the **Columns** tab. Select **Include Columns by Position**. Map `TMODEL_KEY` to 1, `KEY_NAME` to 3, `KEY_VALUE` to 2, and `PARENT_VALUE` to 4.
8. Click **OK** to run the import process.
9. Repeat this process for the `naics-97_tModelKey.txt` and `unspsc-73_tModelKey.txt` files.

---

---

**Note:** If the character set of your database is not UTF-8, do not use the script `iso3166-99.txt` to load the ISO-3166 taxonomy because the taxonomy contains characters from different languages. Instead, use the script `iso3166-99-ascii.txt` to load an ASCII-only version of the taxonomy.

---

---

### Configure OC4J to Use DB2

The following sections describe how to create the DB2 package and modify the URL for regular use.

**Create a DB2 Package** Define a data source with the name and location set to `jdbc/OracleUddi` to reflect that DB2 is the desired database, similar to the following:

```
<data-source
  class="com.evermind.sql.DriverManagerDataSource"
  name="jdbc/OracleUddi"
  location="jdbc/OracleUddi"
  connection-driver="com.oracle.ias.jdbc.db2.DB2Driver"
  username="uddisys"
  password="uddisys"
url="jdbc:oracle:db2://servername:50000;databaseName=UDDISYS;
PackageName=JDBCPKG;DynamicSections=512;
CreateDefaultPackage=TRUE;ReplacePackage=true"
/>
```

Note that *servername* is the network name or IP address of the server hosting the DB2 instance used for the UDDI registry. Also, the line that begins with *url* and the two subsequent lines should be on *one* line, without spaces or returns. They are presented here on three lines for readability.

The data source needs to be made accessible by editing the `data-sources.xml` files in the corresponding `orauddi.ear` and `oraudrepl.ear` applications.

Refer to the Data Sources chapter in the *Oracle Application Server Containers for J2EE Services Guide* for more information.

Now, connect to the UDDI inquiry servlet end point, as shown in the following example, so that these initial URL connection strings will be used to create the appropriate default package in DB2:

```
http://OracleAS-host:port/uddi/inquiry
```

If the request to the inquiry servlet end point hangs or fails, from the DB2 Control Center, check for the JDBCPKGGA and JDBCPKGGB packages under the application objects of the `uddisys` database. If the packages have been created, stop the OC4J instance and proceed with modifying the URL, as described in the next section.

**Modify the URL for Regular Use** Now that the DB2 package has been created, update the data source defined in the previous step (see "[Create a DB2 Package](#)" on page 10-54) and change the URL attribute from:

```
url="jdbc:oracle:db2://servername:50000;databaseName=uddisys;
PackageName=JDBCPKG;DynamicSections=512;
CreateDefaultPackage=TRUE;ReplacePackage=true"
```

to:

```
url="jdbc:oracle:db2://servername:50000;databaseName=uddisys;
PackageName=JDBCPKG;DynamicSections=512"
```

In the preceding examples, the text should be on *one* line, without spaces or returns. They are presented here on multiple lines for readability.

Note that the last two parameters, `CreateDefaultPackage` and `ReplacePackage`, have been removed from the final URL attribute.

Once these changes have been made to the `data-sources.xml` files in the `orauddi.ear` and `oraudrepl.ear` applications, restart the server for the changes to take effect.

Then, connect to the UDDI inquiry servlet end point again, as shown in the following example:

```
http://OracleAS-host:port/uddi/inquiry
```

The OracleAS UDDI Registry page is displayed. You should see the message: "Welcome! Your registry is now up and running."

## Other Oracle Database (Non-Infrastructure)

The following sections describe installation and configuration information for an Oracle database that is not an OracleAS Infrastructure database.

## Script Source Directory

If the `/uddi/admin` directory, located at `${ORACLE_Home}/uddi/admin` on UNIX or `%ORACLE_Home_ORACLE%\uddi\admin` on Windows, is not accessible from the server with the relevant Oracle tools, then copy this directory to a location that is accessible.

## Create the Database and User

Create the `uddisys` database and the `uddisys` user, by taking the following steps:

1. Go to the `${ORACLE_Home}` directory on UNIX or the `%ORACLE_Home_ORACLE%` directory on Windows.
2. Use SQL\*Plus to execute the SQL script `wurinst.sql` using the `sys` user account. For example:

```
sqlplus "sys/change_on_install as sysdba" @wurinst.sql
```

The schema `uddisys` is created with the password `uddisys`. A log file `wurinst.log` is produced.

## Populate the Validated Taxonomy Codes

Populate the validated taxonomy codes using SQL\*Loader with the three control scripts: `naics-97.ctl`, `iso3166-99.ctl`, and `unspsc-73.ctl`. For example:

```
sqlldr userid=uddisys/uddisys control=naics-97.ctl
sqlldr userid=uddisys/uddisys
      control=unspsc-73.ctl
sqlldr userid=uddisys/uddisys control=iso3166-99.ctl
```

---

---

**Note:** If the character set of your database is not UTF-8, do not use the script `iso3166-99.ctl` to load the ISO-3166 taxonomy because the taxonomy contains characters from different languages. Instead, use the script to load an ASCII-only version of the taxonomy:

```
sqlldr userid=uddisys/uddisys control=iso3166-99-ascii.ctl
```

---

---

## Configure OC4J to Use the Non-OracleAS Infrastructure Database

Define a data source with the name and location set to `jdbc/OracleUddi` to reflect that a non-OracleAS Infrastructure database is the desired database, similar to the following:

```
<data-source
  class="oracle.jdbc.pool.OracleConnectionCacheImpl"
  name="jdbc/OracleUddi"
  location="jdbc/OracleUddi"
  connection-driver="oracle.jdbc.driver.OracleDriver"
  username="uddisys"
  password="uddisys"
url="jdbc:oracle:thin:@servername:1521:oracle sid"
/>
```

Note that `servername` is the network name or IP address of the server hosting the non-OracleAS Infrastructure database instance used for the UDDI registry.

The data source needs to be accessible by the `orauddi.ear` and `oraudrepl.ear` applications.

Refer to the Data Sources chapter in the *Oracle Application Server Containers for J2EE Services Guide* for more information.

Restart the UDDI server for these changes to take effect.

## OracleAS UDDI Registry Server Error Messages

The error codes listed are used by UDDI administrators. In general, UDDI error code `E_fatalError` can represent various server-side errors that an administrator has to handle.

The specific server-side error is captured in the J2EE application log file. The log file, `application.log`, for the `orauddi.ear` application is typically located under the `J2EE_HOME/application-deployments/orauddi` directory.

The log file, `application.log`, for the `oraudrepl.ear` application is typically located under the `J2EE_HOME/application-deployments/oraudrepl` directory. The reference provides additional information for an administrator to diagnose and resolve problems.

**WUR-00010: An attempt was made to update a configuration parameter that does not exist "{0}".**

**Cause:** The named UDDI server configuration parameter does not exist.

**Action:** Correct the spelling of the name of the configuration parameter to be updated. Refer to the configuration parameter reference information for details.

**WUR-00011: An attempt was made to update a configuration parameter "{0}" in `uddiserver.config`. That file cannot be found.**

**Cause:** The UDDI server configuration file `uddiserver.config` could not be found.

**Action:** Make sure that the JVM property `oracle.home` of the OC4J instance is defined properly.

**WUR-00012: The specified user name, "{0}", is not a name that is known to the registry.**

**Cause:** The named user does not exist in the registry.

**Action:** Correct the spelling of the named user.

**WUR-00013: The 'Default' role for publishing limits may not be deleted.**

**Cause:** An attempt was made to remove the system-defined user quota role 'Default.'

**Action:** Do not delete the user quota role 'Default.' If the 'Default' user quota role is not desirable, set the quota limits to zero to disable it.

**WUR-00100: An internal error occurred while marshaling the response.**

**Cause:** An unexpected internal error occurred in writing the response to a client.

**Action:** Identify and correct the internal error. The internal error is embedded in the details of the error.

**WUR-00101: An internal error occurred while unmarshaling the request.**

**Cause:** An unexpected internal error occurred in parsing the request sent by a client.

**Action:** Identify and correct the internal error. The internal error is embedded in the details of the error.

**WUR-00104: The value of the configuration parameter named "{0}" is invalid.**

**Cause:** The value of the named UDDI server configuration parameter was invalid.

**Action:** Refer to the configuration parameter reference information for the valid values. Use the UDDI administration tool to update the configuration parameter.

**WUR-00105: A database error with SQL code "{0}" occurred while trying to "{1}".**

**Cause:** An unexpected database error occurred in carrying out the named action.

**Action:** Identify and correct the database error. The database error is embedded in the details of the error.

**WUR-00106: An internal error caused the request to fail to make the specified updates. While rolling back the changes, another error occurred; this leaves data in an unpredictable state.**

**Cause:** An unexpected database error occurred in rollback phases of error processing.

**Action:** Identify and correct the database error. The database error is embedded in the details of the error.

**WUR-00107: An internal error occurred while committing the requested changes to the registry; this leaves data in an unpredictable state.**

**Cause:** An unexpected database error occurred in committing the requested changes.

**Action:** Identify and correct the database error. The database error is embedded in the details of the error.

**WUR-00108: An internal error occurred while trying to get a connection to the underlying database.**

**Cause:** An unexpected database error occurred in obtaining a database connection to serve the request.

**Action:** Identify and correct the database error. The database error is embedded in the details of the error.

**WUR-00109: An internal error occurred while trying to close a connection to the underlying database.**

**Cause:** An unexpected database error occurred during the release of the database connection after the request was served.

**Action:** Identify and correct the database error. The database error is embedded in the details of the error.

**WUR-00110: An internal error occurred while trying to create and set up a data source abstraction for the underlying database.**

**Cause:** An unexpected internal error occurred while creating the database connection pool.

**Action:** Identify and correct the internal error. The internal error is embedded in the details of the error.

**WUR-00111: An internal error occurred while trying to perform a JNDI lookup and locate of the object "{0}".**

**Cause:** An internal error occurred in obtaining the named object from the JNDI context. Examples of possible objects include database connection pools, message queues, and so forth.

**Action:** Identify and correct the internal error. The internal error is embedded in the details of the error.

**WUR-00113: An internal error occurred while trying to access the repository API to set up a data source abstraction.**

**Cause:** An unexpected internal error occurred while creating the database connection pool using Oracle Application Server metadata repository access API.

**Action:** Identify and correct the internal error. The internal error is embedded in the details of the error.

**WUR-00114: An internal error occurred while trying to generate a Universal Unique Identifier (UUID).**

**Cause:** An unexpected internal error occurred while generating a UUID.

**Action:** Identify and correct the internal error. The internal error is embedded in the details of the error.

**WUR-00115: The registry was unable to retrieve OC4J-specific environment settings from the J2EE container; the user "{0}" cannot be authenticated.**

**Cause:** An unexpected internal error occurred while authenticating the user. The error is usually due to incorrect settings in web.xml or using an unsupported version of the OC4J container.

**Action:** Identify and correct the internal error. The internal error is embedded in the details of the error.

**WUR-00116: An internal error occurred while performing the automatic postinstallation configuration for the UDDI registry. Regular registry operations cannot proceed if the registry is not properly configured.**

**Cause:** An unexpected internal error occurred in performing the automatic postinstallation configuration for the UDDI registry.

**Action:** Identify and correct the internal error. The internal error is embedded in the details of the error.

**WUR-00117: Cannot close data source properly.**

**Cause:** An unexpected internal error occurred while closing the database connection pool during shutdown of the UDDI registry.

**Action:** Identify and correct the internal error. The internal error is embedded in the details of the error.

**WUR-00200: An internal error occurred during external validation.**

**Cause:** An unexpected internal error occurred while making a validation call to an external validation service.

**Action:** Identify and correct the internal error. The internal error is embedded in the details of the error.

**WUR-00201: An internal error occurred during external validation while processing the in-memory request.**

**Cause:** An unexpected internal error occurred while processing the UDDI entities in the request before they were sent for external validation.

**Action:** Identify and correct the internal error. The internal error is embedded in the details of the error.

**WUR-00202: An internal error occurred during external validation because the tModel list property, "{0}", has the wrong format.**

**Cause:** The value of the UDDI server configuration property, oracle.uddi.server.externalValidationTModelList, was invalid.

**Action:** Correct the value. Refer to the configuration parameter reference information for details.

**WUR-00203: An internal error occurred during external validation because the timeout property, "{0}", is not the right integer format.**

**Cause:** The value of the UDDI server configuration property, oracle.uddi.server.externalValidationTimeout, was invalid.

**Action:** Correct the value. Refer to the configuration parameter reference information for details.

**WUR-00204: An internal error occurred during external validation because the response is not a correct DispositionReport.**

**Cause:** DispositionReport returned by the external validation service was invalid. For example, DispositionReport was empty.

**Action:** Contact the external validation service provider.

**WUR-00205: An internal error occurred during external validation because the response is not expected. The response is of code "{0}" with message "{1}".**

**Cause:** DispositionReport returned by the external validation service contained an unexpected DispositionReport error number.

**Action:** Contact the external validation service provider.

**WUR-00300: DB schema version is missing. Please check DB for VERSION table.**

**Cause:** The version of the database schema for persistent storage was missing.

**Action:** Contact Oracle Support Services.

**WUR-00301: DB schema version "{0}" is incompatible with mid-tier version. DB schema must be updated to make the UDDI registry function.**

**Cause:** The version of the database schema for persistent storage was not supported by the version of the registry being used.

**Action:** Upgrade the database schema to the latest version. Refer to the UDDI database schema upgrade documentation for details.

**WUR-00302: An internal error occurred while trying to retrieve and load the UDDI DELTA server property file.**

**Cause:** An internal error occurred while initializing the UDDI registry in the backward compatibility mode with an older version of the database schema.

**Action:** Contact Oracle Support Services.

**WUR-00303: This operation is not allowed by DB schema version "{0}". You must upgrade DB schema to the latest version to carry out this operation.**

**Cause:** The requested operation was not supported because the UDDI registry was running in the backward compatibility mode with an older version of the database schema.

**Action:** Upgrade the database schema to the latest version. Refer to the UDDI database schema upgrade documentation for details.

**WUR-05001: Cannot find the UDDI entity just saved.**

**Cause:** An unexpected internal error occurred in updating the update journal.

**Action:** Contact Oracle Support Services.

**WUR-05002: Cannot perform custody transfer for an entity that is not businessEntity or tModel. The key of the offending entity is "{0}".**



**Cause:** In the custody transfer change record, the specified UDDI entity is not businessEntity or tModel.

**Action:** Contact the administrator of the UDDI node where the change record originated.

**WUR-05003: Warning: Received a duplicate change record originating from node "{0}" with usn "{1}".**

**Cause:** A duplicate change record sent from the named UDDI node was detected.

**Action:** No action is needed. This is merely an informational message.

**WUR-05004: Received an out-of-order change record originating from node "{0}" with usn "{1}". The change record with usn "{2}" has been processed.**

**Cause:** The named change record was received after a change record with a larger update sequence number (USN) had been processed.

**Action:** Contact the administrator of the UDDI node where the change record originated.

**WUR-05005: The change record originating from node "{0}" with usn "{1}" is invalid because the named node is not recognized.**

**Cause:** The originating node of the named change record was not recognized. In other words, the node was not recorded in the replication communication graph.

**Action:** Contact the administrator of the UDDI node that provided the change record.

## Command-Line Options for the uddiadmin.jar Tool

The following sections describe the options for the `uddiadmin.jar` command-line tool. In most cases, the command line uses the following format (`setWalletPassword` uses a different URL):

```
java -jar uddiadmin.jar http://OracleAS-host:port/uddi/admin username password  
[-verbose] options_and_their_parameters
```

Make sure that you enter the command on one line. For more information about the `uddiadmin.jar` command-line tool, see ["Using the Command-Line Tool uddiadmin.jar"](#) on page 10-34.

### changeOwner

**Format**

```
-changeOwner new_username {-businessKey business_Key | -tModelKey  
tModel_Key}
```

**Description**

Changes the ownership of the named entity to the new specified user.

### correctChangeRecord

**Format**

```
-correctChangeRecord changeRecordCorrectionfile changeRecord_  
NewDatafile
```

**Description**

Applies the `changeRecordCorrectionfile` file contents and `changeRecordNewDatafile` file contents to the UDDI node. The content of these files must conform to the UDDI replication XML schema. This option is part of UDDI replication error recovery.

### deleteEntity

**Format**

```
-deleteEntity {-businessKey business_Key | -serviceKey  
serviceKey | -bindingKey binding_Key | -tModelKey tModel_Key}
```

**Description**

Deletes the named entity irrespective of the owner of the entity. Note that this operation performs a nonpermanent delete (hide) operation in the case of a `tModel` entity.

### deleteRoleQuotaLimits

**Format**

```
-deleteRoleQuotaLimits role_Name [role_Name ...]
```

**Description**

Deletes the group-to-quota-limit mappings for the specified quota groups. See ["Deleting a Quota Group \(Advanced Operation\)"](#) on page 10-39 for information on using this option.

**Example**

```
java -jar uddiadmin.jar http://OracleAS-host:port/uddi/admin username password
-deleteRoleQuotaLimits role_Name
```

**destroyTModel****Format**

```
-destroyTModel tModel_Key
```

**Description**

Permanently deletes the named tModel from the registry (as opposed to the UDDI-defined delete\_tModel call, which is just hiding the tModel entity).

**doPing****Format**

```
-doPing replicationEndPointUrl [-timeout timeInMilliseconds ][
-walletPassword wallet_password]
```

**Description**

Sends a UDDI replication do\_ping message to the replication end-point URL specified. This is similar to the ping command in TCP/IP that is used to check if the other end point is active. The optional walletPassword parameter is useful when the JVM, which receives the do\_ping message, does not have a valid wallet password set.

**Example**

```
java -jar uddiadmin.jar http://OracleAS-host:port/uddi/admin username password
-doPing http://OracleAS-host:port/uddirepl/replication
```

**downloadReplicationConfiguration****Format**

```
-downloadReplicationConfiguration
```

**Description**

Downloads the currently used replication configuration from a specified UDDI node within OracleAS UDDI Registry. You must upload a replication configuration before you can successfully download one. See ["Enabling UDDI Replication"](#) on page 10-41 for information about using this option.

**Example**

```
java -jar uddiadmin.jar http://OracleAS-host:port/uddi/admin username password
-downloadReplicationConfiguration
```

**getChangeRecord****Format**

```
-getChangeRecord local_usn
```

**Description**

Gets the detail of a change record specified by local\_usn (an integer). This API is used in conjunction with the CorrectChangeRecord option to correct wrong or inconsistent data across different UDDI nodes with OracleAS UDDI Registry.

## getHighWaterMarks

**Format**

```
-getHighWaterMarks replicationEndPointUrl [ -walletPassword  
wallet_password]
```

**Description**

Gets the high-water marks vector from the UDDI node specified by the *replicationEndPointUrl* parameter. The optional *walletPassword* parameter is useful when the JVM, which receives the *do\_ping* message, does not have a valid wallet password set.

**Example**

```
java -jar uddiadmin.jar http://OracleAS-host:port/uddi/admin username password  
-getHighWaterMarks http://OracleAS-host:port/uddirepl/replication
```

## getProperties

**Format**

```
-getProperties
```

**Description**

Lists the current registry configuration parameters. See ["Configuring the Server"](#) on page 10-35 for information about using this option.

**Example**

```
java -jar uddiadmin.jar http://OracleAS-host:port/uddi/admin username password  
-getProperties
```

## getRoleQuotaLimits

**Format**

```
-getRoleQuotaLimits
```

**Description**

Displays all the J2EE-role-to-quota-limits mappings that are currently set in the registry. See ["Viewing the Lists of Quota Groups and Their Limits"](#) on page 10-39 for information about using this option.

**Example**

```
java -jar uddiadmin.jar http://OracleAS-host:port/uddi/admin username password  
-getRoleQuotaLimits
```

## getUserDetail

**Format**

```
-getUserDetail username_to_retrieve
```

**Description**

Retrieves the details of the named user, currently the *authorizedName* of each user. See ["Managing Users"](#) on page 10-36 for information about using this option.

**Example**

```
java -jar uddiadmin.jar http://OracleAS-host:port/uddi/admin username password  
-getUserDetail username
```

## getUsers

### Format

```
-getUsers
```

### Description

Lists all existing users who have entities in the registry. See ["Managing Users"](#) on page 10-36 for information about using this option.

### Example

```
java -jar uddiadmin.jar http://OracleAS-host:port/uddi/admin username password
[-verbose] -getUsers
```

## import

### Format

```
-import [-s|-m]
    -businesses filename |
    -tmodels filename    |
    -assertions filename -fromBusinessCheck {true|false}
                        -toBusinessCheck {true|false} ]
```

### Description

Imports all businessEntity and tModel data structures, and a publisherAssertion data structure in the named file. For importing the businessEntity data structure, the named file (filename) for importing should contain a UDDI businessDetail XML document. For importing tModel data structures, the named file should contain a UDDI tModelDetail XML document. By importing them, entity keys (such as businessKey, serviceKey, bindingKey, tModelKey) are preserved. The operatorName and authorizedName fields, however, are not preserved. The operatorName field will be replaced by the operatorName configuration parameter of the registry. The owner of the imported entities is the administrator; hence, the authorizedName field will be the authorized name of the administrator.

Importing can be done in single mode (-s), which does not allow partial success (some entities are imported and some are not due to some error condition), or in multiple mode (-m), which does allow partial success.

The import parameter is particularly useful in importing the well-known service interface specification tModel and classification tModel data structures from some authoritative sources.

Because the entity keys are preserved, administrators should be careful in evaluating the source of the entities to ensure there will not be a collision in entity keys.

For importing a publisherAssertion, two Boolean values are required. These Boolean values are used to indicate from which side (or both sides when two Boolean values are true) the publisherAssertion is going to be inserted.

See ["Importing Entities"](#) on page 10-40 for information about using this option.

### Example

The following example imports the publisherAssertion contained in the file assert.xml:

```
java -jar uddiadmin.jar http://OracleAS-host:port/uddi/admin username password
-import -s -assertions assert.xml -fromBusinessCheck true
```

```
-toBusinessCheck true
```

## setOperationalInfo

### Format 1

```
-setOperationalInfo {-businessKey key | -tModelKey key}  
[-newOperator OperatorName] [-newAuthorizedname authName]  
[-newTime timestamp]
```

### Format 2

```
-setOperationalInfo {-serviceKey key | -bindingKey key}  
[-newTime timestamp]
```

### Description

Sets some operational information, such as the operator name, authorized name, or timestamp of a businessEntity or tModel data structure specified by a key, for example, following an import operation. You can set any combination of operator name, authorized name, or timestamp using the `setOperationalInfo` option.

Format 1 lets you change either the operator name, the authorized name, or the timestamp, or all three, of the businessEntity or tModel specified by a key.

Format 2 lets you change only the timestamp of a businessService or bindingTemplate.

---

---

**Note:** The format of a timestamp is defined as 'yyyy-mm-dd hh.mm:ss.fffffff' by `java.sql.Timestamp`. For example:

```
'2002-12-01 00:00:00'
```

Because there is a blank space in the timestamp value between 'yyyy-mm-dd' and 'hh.mm:ss.fffffff', the entire value must be placed inside a pair of single quotation marks on the command line.

---

---

---

---

**Caution:** In general, the `setOperationalInfo` option should not be used when replication is enabled.

---

---

See "[Setting Operational Information](#)" on page 10-41 for information about using this option.

## setProperty

### Format

```
-setProperty property_name=value
```

### Description

Changes the value of the named server configuration property. The OracleAS UDDI Registry J2EE application needs to be restarted for the changes to take effect.

### Example

The following example sets the `operatorName` property to `OracleUddiServerIT_Dept`:

```
java -jar uddiadmin.jar http://OracleAS-host:port/uddi/admin username password  
-setProperty oracle.uddi.server.operatorName=OracleUddiServerIT_Dept
```

---



---

**Caution:** Be very careful when using the `setProperty` option to change the value of server configuration properties. Setting an incorrect value for a property could cause severe damage to the integrity of the registry.

---



---

See "[Modifying Properties at Installation or First-Use](#)" on page 10-7 for more information on this option.

## setRoleQuotaLimits

### Format

```
-setRoleQuotaLimits roleName [maxBE] [maxBSperBE] [maxBTperBS]
[maxTM] [maxPA]
```

### Description

Sets the quota limit value for the specified quota group. This option can be used to create a new group-to-quota-limit mapping or to update an existing mapping. The parameters are defined as follows:

- `roleName`—name of the quota group to map to the specified limits
- `maxBE`—maximum number of `businessEntity` data structures allowed
- `maxBSperBE`—maximum number of `businessService` data structures per `businessEntity` allowed
- `maxBTperBS`—maximum number of `bindingTemplate` data structures per `businessEntity` allowed
- `maxTM`—maximum number of `tModel` data structures allowed
- `maxPA`—maximum number of `publisherAssertion` data structures allowed

The value `-1` means unlimited.

See "[Updating the Limits of a Quota Group](#)" on page 10-38 for more information about this option.

### Example

```
java -jar uddiadmin.jar http://OracleAS-host:port/uddi/admin username password
-setRoleQuotaLimits roleName maxBE maxBSperBE maxBTperBS maxTM maxPA
```

## setWalletPassword

### Format

```
-setWalletPassword wallet_password
```

### Description

Sets the wallet password to be used for HTTPS communication among UDDI nodes for UDDI replication. Each time the application is restarted, this option must be invoked because the wallet password is not stored persistently, for security reasons. The registry replication wallet admin URL is:

```
http://OracleAS-host:port/uddirepl/admin/wallet
```

See "[Enabling UDDI Replication](#)" on page 10-41 for more information about using this option with UDDI replication.

**Example**

```
java -jar uddiadmin.jar http://OracleAS-host:port/uddirepl/admin/wallet username  
password  
-setWalletPassword=walletpassword
```

**transferCustody**

**Format**

```
-transferCustody oldOperatorName newOperatorName  
newAuthorizedName {-tModelKey tModel_Key | -businessKey  
businessKey}
```

**Description**

Transfers the custody of a tModel or a businessEntity to a new operator and a new authorized name. This option is part of custody transfer as defined by the UDDI specification.

**uploadReplicationConfiguration**

**Format**

```
-uploadReplicationConfiguration xml_file_containing_replication_  
configuration
```

**Description**

Uploads the specified replication configuration to a particular UDDI node within OracleAS UDDI Registry. The application must be restarted for the new replication configuration to be used. See ["Enabling UDDI Replication"](#) on page 10-41 for information about using this option.

**Example**

```
java -jar uddiadmin.jar http://OracleAS-host:port/uddi/admin username password  
-uploadReplicationConfiguration xml_file_containing_replication_configuration
```



---

## Server Configuration Properties

This section provides reference information for UDDI server configuration properties. The properties are of the class `oracle.uddi.server` and its subclasses. You set them by using the options of the command-line tool `uddiadmin.jar`. See ["Using the Command-Line Tool `uddiadmin.jar`"](#) on page 10-34 for more information about the `uddiadmin.jar` command-line tool.

### `addressTModelKeyValidation`

**Description**

Determines if `tModelKey` existence validation occurs within address elements.

**Property Type/Allowable Values**

Boolean (true, false)

**Initial Value**

true

**Typical Value**

true

**Example**

```
java -jar uddiadmin.jar http://OracleAS-host:port/uddi/admin username password
-setProperty oracle.uddi.server.addressTModelKeyValidation=true
```

### `assertionKeyedRefValidation`

**Description**

Controls validation for all `publisherAssertion keyedReference` entities.

**Property Type/Allowable Values**

- `full`: All validation conditions will be checked.
- `tmodel_existence`: Only `tModelKey` existence will be checked.
- `none`: No condition will be checked.

**Initial Value**

full

**Typical Value**

full

**Example**

```
java -jar uddiadmin.jar http://OracleAS-host:port/uddi/admin username password
-setProperty oracle.uddi.server.assertionKeyedRefValidation=full
```

### `businessEntityURLPrefix`

**Description**

Provides the prefix of the generated discovery URL, which is automatically generated for each `businessEntity` data structure saved in the registry. The prefix should be customized for your deployment environment. Setting this parameter applies in a retroactive fashion to existing entities in the database. For example, changing the

discoveryURL prefix results in all discovery URLs of usetype businessEntity that begin with the old URL prefix to be changed to the new URL prefix.

**Property Type/Allowable Values**

A valid URL.

**Initial Value**

OracleAS UDDI Registry generates an initial value upon server initialization.

**Typical Value**

The host name and port should be the host name and port of the Web server (which may or may not be the same as the servlet container).

**Notes**

See "[Modifying Properties at Installation or First-Use](#)" on page 10-7 for information about using this property.

**Example**

```
java -jar uddiadmin.jar http://OracleAS-host:port/uddi/admin username password
-setProperty
oracle.uddi.server.businessEntityURLPrefix="http://uddihost:port/uddi/inquiryget"
```

## categoryValidation

**Description**

Controls validation for all CategoryBag entities.

**Property Type/Allowable Values**

- full: All validation conditions will be checked.
- tmodel\_existence: Only tModelKey existence will be checked.
- none: No condition will be checked.

**Initial Value**

full

**Typical Value**

full

**Example**

```
java -jar uddiadmin.jar http://OracleAS-host:port/uddi/admin username password
-setProperty oracle.uddi.server.categoryValidation=full
```

## categoryValidationTModelKeys

**Description**

Represents the categorization and identifier tModel keys, which will be validated by the registry during an attempted save operation.

**Property Type/Allowable Values**

A list in the form of '<tModelKey1>', '<tModelKey2>', '<tModelKey3>'.

**Initial Value**

'UUID:C1ACF26D-9672-4404-9D70-39B756E62AB4', which represents the uddi-org:types classification. The preinstalled value, however, is the UDDI types classification plus the three classifications defined in the UDDI v1.0 specification: (uddi-org:types, uddi-org:iso-ch:3166-1999, ntis-gov:naics:1997, unspsc-org:unspsc).

**Typical Value**

The preinstalled value.

**Notes**

The `uddi-org:types` classification *should not* be removed from the list. In addition, you must enter the command on one line, with no returns or no extra spaces.

**Example**

```
java -jar uddiadmin.jar http://OracleAS-host:port/uddi/admin username password
-setProperty
"oracle.uddi.server.categoryValidationTModelKeys=
'UUID:C1ACF26D-9672-4404-9D70-39B756E62AB4',
'UUID:4E49A8D6-D5A2-4FC2-93A0-0411D8D19E88',
'UUID:C0B9FE13-179F-413D-8A5B-5004DB8E5BB2',
'UUID:CD153257-086A-4237-B336-6BDCBDC6634' "
```

**changeRecordWantsAck****Description**

Controls whether or not ACK is required for the change records sent out from the local node.

**Property Type/Allowable Values**

Boolean (true, false)

**Initial Value**

false

**Typical Value**

false

**Example**

```
java -jar uddiadmin.jar http://OracleAS-host:port/uddi/admin username password
-setProperty oracle.uddi.server.replication.changeRecordWantsAck=false
```

**defaultLang****Description**

Provides the default language of the registry for the purpose of filling in UDDI v1.0 description elements, which lack a language qualification. Language defaults are not done for UDDI v2 requests. Valid values are the values of the `xml:lang` attribute.

**Property Type/Allowable Values**

Values of `xml:lang`.

**Initial Value**

en

**Typical Value**

The location of the primary region the registry serves.

**Example**

```
java -jar uddiadmin.jar http://OracleAS-host:port/uddi/admin username password
-setProperty oracle.uddi.server.defaultLang=en
```

## externalValidation

### Description

Determines if external validation occurs.

### Property Type/Allowable Values

Boolean (true, false)

### Initial Value

false

### Typical Value

false

### Notes

See "[Enabling External Category Validation](#)" on page 10-47 for information on using this property.

### Example

```
java -jar uddiadmin.jar http://OracleAS-host:port/uddi/admin username password
-setProperty oracle.uddi.server.externalValidation=true
```

## externalValidationTimeout

### Description

Defines the amount of time, in milliseconds, before a timeout occurs for external validation.

### Property Type/Allowable Values

long

### Initial Value

5000

### Typical Value

NA

### Notes

See "[Adding an Externally Validated Category to the Registry](#)" on page 10-47 for information on using this property.

### Example

```
java -jar uddiadmin.jar http://OracleAS-host:port/uddi/admin username password
-setProperty oracle.uddi.server.externalValidationTimeout=5000
```

## externalValidationTModelList

### Description

Provides the list of tModel key-URL pairs that represents the category and identifier tModel data structures that will be validated by an external SOAP service. The tModelKey and URL values within a pair are separated by a comma (,), and pairs of values are separated by a semicolon (;).

### Property Type/Allowable Values

NA

**Initial Value**

null value ""

**Typical Value**

null value ""

**Notes**

See ["Adding an Externally Validated Category to the Registry"](#) on page 10-47 for information on using this property.

**Example**

```
java -jar uddiadmin.jar http://OracleAS-host:port/uddi/admin username password
-setProperty
oracle.uddi.server.externalValidationTModelList=uuid:acme-taxonomy-key,
http://acme.com/externalValidation
```

**hostingRedirectorValidation****Description**

Determines if hostingRedirector validation occurs within bindingTemplate elements. Validation ensures that the referenced bindingTemplate element exists and does not contain a hostingRedirector element.

**Property Type/Allowable Values**

Boolean (true, false)

**Initial Value**

true

**Typical Value**

true

**Example**

```
java -jar uddiadmin.jar http://OracleAS-host:port/uddi/admin username password
-setProperty oracle.uddi.server.hostingRedirectorValidation=true
```

**identifierValidation****Description**

Controls validation for all IdentifierBag entities.

**Property Type/Allowable Values**

- full: All validation conditions will be checked.
- tmodel\_existence: Only tModelKey existence will be validated.
- none: No condition will be checked.

**Initial Value**

full

**Typical Value**

full

**Example**

```
java -jar uddiadmin.jar http://OracleAS-host:port/uddi/admin username password
-setProperty oracle.uddi.server.identifierValidation=full
```

## **jdbcDriverType**

**Description**

Defines the type of JDBC driver to be used to access the OracleAS Infrastructure database. This property is applicable only if the OracleAS Infrastructure database is used as the back-end storage.

**Property Type/Allowable Values**

thin or oci

**Initial Value**

thin

**Typical Value**

NA

**Notes**

In a cluster environment, this property must be set for each OC4J instance.

**Example**

```
java -jar uddiadmin.jar http://OracleAS-host:port/uddi/admin username password
-setProperty oracle.uddi.server.db.jdbcDriverType=thin
```

## **maxChangeRecordsSentEachTime**

**Description**

Controls the maximum number of change records sent out in response to an incoming `getChangeRecords` request.

**Property Type/Allowable Values**

integer

**Initial Value**

100

**Typical Value**

NA

**Example**

```
java -jar uddiadmin.jar http://OracleAS-host:port/uddi/admin username password
-setProperty oracle.uddi.server.replication.maxChangeRecordsSentEachTime=100
```

## **maxConnections**

**Description**

Determines the maximum number of database connections in the connection pool. This property is applicable only if the OracleAS Infrastructure database is used as the back-end storage.

**Property Type/Allowable Values**

A positive integer.

**Initial Value**

8

**Typical Value**

Depends on the maximum number of concurrent requests and the desired performance.

**Notes**

Enter a value that is the estimated maximum number of concurrent requests plus a percentage of the buffer.

In a cluster environment, this property must be set for each OC4J instance.

**Example**

```
java -jar uddiadmin.jar http://OracleAS-host:port/uddi/admin username password  
-setProperty oracle.uddi.server.db.maxConnections=10
```

**minConnections****Description**

Determines the minimum number of database connections in the connection pool. This property is applicable only if the OracleAS Infrastructure database is used as the back-end storage.

**Property Type/Allowable Values**

A nonnegative integer that is smaller than the value for `maxConnections`.

**Initial Value**

1

**Typical Value**

1

**Notes**

In a cluster environment, this property must be set for each OC4J instance.

**Example**

```
java -jar uddiadmin.jar http://OracleAS-host:port/uddi/admin username password  
-setProperty oracle.uddi.server.db.minConnections=1
```

**operatorCategory****Description**

Determines whether or not additional entities may be categorized as an operator node, if the value of the `categoryValidation` property is true.

**Property Type/Allowable Values**

Boolean (true, false)

**Initial Value**

true

**Typical Value**

true

**Example**

```
java -jar uddiadmin.jar http://OracleAS-host:port/uddi/admin username password  
-setProperty oracle.uddi.server.categoryValidation.operatorCategory=true
```

## operatorName

**Description**

Provides the name of the operator of OracleAS UDDI Registry. This name appears in the operator attribute of responses. Setting this parameter applies in a retroactive fashion to existing entities in the database. For example, changing the operator name results in all business and tModel data structures that currently have the old operator name to have that name changed to the new operator name.

**Property Type/Allowable Values**

A non-null string.

**Initial Value**

OracleUddiServer

**Typical Value**

*domain\_of\_the\_UDDI\_registry/uddi*

**Notes**

Be sure to set this parameter before enabling replication.

See ["Adding an Externally Validated Category to the Registry"](#) on page 10-47 for information on using this property.

**Example**

```
java -jar uddiadmin.jar http://OracleAS-host:port/uddi/admin username password
  -setProperty oracle.uddi.server.operatorName=OracleUddiServerIT_Dept
```

## pushEnabled

**Description**

Controls whether or not a push task should be performed for UDDI replication.

**Property Type/Allowable Values**

Boolean (true, false)

**Initial Value**

true

**Typical Value**

true

**Example**

```
java -jar uddiadmin.jar http://OracleAS-host:port/uddi/admin username password
  -setProperty oracle.uddi.server.replication.pushEnabled=true
```

## pushTaskExecutionPeriod

**Description**

Controls the push task execution period (in milliseconds).

**Property Type/Allowable Values**

45000

**Initial Value**

NA



**Typical Value**

NA

**Example**

```
java -jar uddiadmin.jar http://OracleAS-host:port/uddi/admin username password
-setProperty oracle.uddi.server.replication.pushTaskExecutionPeriod=45000
```

**quotaLimitChecking****Description**

Determines whether or not publishing quotas, the limits on the number of entities that can be created in the registry per user, are enforced.

**Property Type/Allowable Values**

Boolean (true, false)

**Initial Value**

true

**Typical Value**

true

**Notes**

See "[Enforcing Quotas](#)" on page 10-37 for information on quota limits.

**Example**

```
java -jar uddiadmin.jar http://OracleAS-host:port/uddi/admin username password
-setProperty oracle.uddi.server.quotaLimitChecking=true
```

**schemaValidationUponIncomingRequests****Description**

Determines whether or not the server will validate incoming requests against the UDDI XML schema.

**Property Type/Allowable Values**

Boolean (true, false)

**Initial Value**

true

**Typical Value**

true

**Example**

```
java -jar uddiadmin.jar http://OracleAS-host:port/uddi/admin username password
-setProperty oracle.uddi.server.schemaValidationUponIncomingRequests=true
```

**soapRequestAuthMethod****Description**

Controls the authentication method the registry node will try to use in sending replication SOAP requests to other nodes. If the value CLIENT-CERT is used, the administrator must set the wallet password each time the registry node gets started or restarted.

**Property Type/Allowable Values**

NONE or CLIENT-CERT

**Initial Value**

NONE

**Typical Value**

CLIENT-CERT

**Example**

```
java -jar uddiadmin.jar http://OracleAS-host:port/uddi/admin username password  
-setProperty oracle.uddi.server.replication.soapRequestAuthMethod=NONE
```

## soapRequestTimeout

**Description**

Controls the timeout value for each SOAP replication request (in milliseconds).

**Property Type/Allowable Values**

long

**Initial Value**

180000

**Typical Value**

NA

**Example**

```
java -jar uddiadmin.jar http://OracleAS-host:port/uddi/admin username password  
-setProperty oracle.uddi.server.replication.soapRequestTimeout=180000
```

## startMaintainingUpdateJournal

**Description**

Controls whether or not the update journal will be maintained for UDDI replication. This property must be set to true for replication to occur.

**Property Type/Allowable Values**

Boolean (true, false)

**Initial Value**

false

**Typical Value**

true

**Notes**

Be sure to upload a correct replication configuration before you set this property to true.

Once you set this property to true, you should set it back to false only if you no longer want to participate in UDDI replication. Setting this property haphazardly from true to false will result in fatal loss of change records.

See "[Enabling UDDI Replication](#)" on page 10-41 for information on using this property.

**Example**

```
java -jar uddiadmin.jar http://OracleAS-host:port/uddi/admin username password
```

```
-setProperty oracle.uddi.server.replication.startMaintainingUpdateJournal=false
```

## status

### Description

Indicates whether or not the scheduler is enabled to send out replication requests.

### Property Type/Allowable Values

Boolean (0=off, 1=on)

### Initial Value

1

### Typical Value

1

### Notes

See "[Enabling UDDI Replication](#)" on page 10-41 for information on using this property.

### Example

```
java -jar uddiadmin.jar http://OracleAS-host:port/uddi/admin username password  
-setProperty oracle.uddi.server.scheduler.status=1
```

## stmtCacheType

### Description

Defines the type of statement caching. This property is to be used with the OracleAS Infrastructure database and JDBC driver only.

### Property Type/Allowable Values

NONE, IMPLICIT, or EXPLICIT

### Initial Value

NONE

### Typical Value

EXPLICIT

### Notes

In a cluster environment, this property must be set for each OC4J instance.

### Example

```
java -jar uddiadmin.jar http://OracleAS-host:port/uddi/admin username password  
-setProperty oracle.uddi.server.db.stmtCacheType=NONE
```

## stmtCacheSize

### Description

Defines the size (number of statements cached) of statement caching per connection. This property is to be used with the OracleAS Infrastructure database and JDBC driver only.

### Property Type/Allowable Values

integer

### Initial Value

50

**Typical Value**

50

**Example**

```
java -jar uddiadmin.jar http://OracleAS-host:port/uddi/admin username password  
-setProperty oracle.uddi.server.db.stmtCacheSize=50
```

## taskExecutionPeriod

**Description**

Controls the period of time during which a replication task should be executed (in milliseconds).

**Property Type/Allowable Values**

long

**Initial Value**

5000

**Typical Value**

NA

**Example**

```
java -jar uddiadmin.jar http://OracleAS-host:port/uddi/admin username password  
-setProperty oracle.uddi.server.replication.taskExecutionPeriod=5000
```

## timer\_pool\_size

**Description**

Specifies the number of concurrently active threads used by the scheduler.

**Property Type/Allowable Values**

NA

**Initial Value**

1

**Typical Value**

1

**Notes**

See "[Setting Properties for the UDDI Replication Scheduler](#)" on page 10-43 for information on using this property.

**Example**

```
java -jar uddiadmin.jar http://OracleAS-host:port/uddi/admin username password  
-setProperty oracle.uddi.server.scheduler.timer_pool_size=1
```

## tModelInstanceInfoKeyValidation

**Description**

Determines if tModelKey existence validation occurs within tModelInstanceInfo elements.

**Property Type/Allowable Values**

Boolean (true, false)

**Initial Value**

true

**Typical Value**

true

**Example**

```
java -jar uddiadmin.jar http://OracleAS-host:port/uddi/admin username password  
-setProperty oracle.uddi.server.tModelInstanceInfoKeyValidation=true
```

**walletLocation****Description**

Defines the wallet file name. The wallet file will be located in the same place as the `uddiserver.config` file.

**Property Type/Allowable Values**

NA

**Initial Value**

ewallet.p12

**Typical Value**

NA

**Notes**

See "[Enabling UDDI Replication](#)" on page 10-41 for information on using this property.

**Example**

```
java -jar uddiadmin.jar http://OracleAS-host:port/uddi/admin username password  
-setProperty oracle.uddi.server.replication.walletLocation=ewallet.p12
```



---



---

## Consuming Web Services in J2EE Applications

This chapter describes how to consume Web Services in Java 2 Platform, Enterprise Edition (J2EE) applications. One type of Web-based information or services is supported:

- SOAP-based Web Services described using WSDL, see [Consuming SOAP-Based Web Services Using WSDL](#).

In addition, when a J2EE application acquires a WSDL document at runtime, the dynamic invocation API is used to invoke any SOAP operation described in the WSDL document. See [Dynamic Invocation of Web Services](#) for information about how to use the dynamic invocation API.

### Consuming SOAP-Based Web Services Using WSDL

The `wsdl2ejb` utility can be used by J2EE developers to consume a Web Service described in Web Services Description Language (WSDL) document into their applications. This utility takes a WSDL document and some additional optional parameters and produces an EJB EAR file that can be deployed into OC4J. The EJB Remote Interface is generated based on the WSDL portType. Each WSDL operation is mapped to an EJB method. The EJB method parameters are derived from the WSDL operation input message parts, while the EJB method return value is mapped from the parts of the WSDL operation output message. The Oracle SOAP Mapping Registry is used to map XML types to the corresponding Java types.

Additional references regarding WSDL and SOAP can be found in the following locations:

- The WSDL 1.1 specification is available at  
<http://www.w3.org/TR/wsdl>
- The SOAP 1.1 specification is available at  
<http://www.w3.org/TR/SOAP/>

The command-line options for running the `wsdl2ejb` utility are described in [Table 11-1](#).

**Table 11-1** *wsdl2ejb Utility Command-Line Options*

Option	Description
<code>-conf &lt;config file&gt;</code>	Allows the <code>wsdl2ejb</code> utility to load a configuration file.

**Table 11–1 (Cont.) wsdl2ejb Utility Command-Line Options**

Option	Description
-d <i>&lt;destDir&gt;</i>	Allows a destination directory to be specified where the generated EJB EAR file is to be written.
-Dhttp.proxyHost	Allows the proxy host name to be specified when an HTTP URL is used to supply the location of the WSDL document and an HTTP proxy server is required to access it.
-Dhttp.proxyPort	Allows the proxy port number to be specified when an HTTP URL is used to supply the location of the WSDL document and an HTTP proxy server is required to access it.
-jar	Allows you to specify the <code>wsdl2ejb</code> utility as a JAR file.

To run the `wsdl2ejb` utility, enter the following command where *<destDir>* is the destination directory to where the generated EJB EAR file is to be written and the file `mydoc.wsdl` is the location of the WSDL document:

```
java -jar wsdl2ejb.jar -d <destDir> mydoc.wsdl
```

---

**Note:** The `wsdl2ejb.jar` file is located in your `$ORACLE_HOME/webservices/lib` installation directory for UNIX or `%ORACLE_HOME%\webservices\lib` installation directory for Windows.

---

If an HTTP URL is used to supply the location of the WSDL document and an HTTP proxy is required to access it, the following command and syntax must be used to run the utility:

```
java -Dhttp.ProxyHost=myProxyHost -Dhttp.proxyPort=80 -jar wsdl2ejb.jar -d
<destDir> http://myhost/mydoc.wsdl
```

In this example, the utility uses the supplied WSDL to generate the EJB EAR file in the destination directory (*<destDir>*). The EJB class name, Java Naming and Directory Interface (JNDI) binding key, and Java package name are derived from the location of the SOAP service described in the WSDL.

In this command syntax, the `wsdl2ejb` utility maps the XML types, which are supported by default by the Oracle SOAP Mapping Registry.

The `wsdl2ejb` utility generates the following sets of files located within the destination directory name (*<destDir>*) that you specify in the command line. The utility saves the generated files using the following directory layout:

```
Root /
+ app.ear
+ src/
+ ... generated java sources ...
+ classes/
+ META-INF/
+ ejb-jar.xml
+ ... compiled classes and xml resources ....
+ deploy/
+ ejb.jar
+ META-INF/
+ application.xml
```



- An .ear file (which is a JAR archive containing the J2EE application that can be deployed in OC4J) is located within the destination directory (*<destDir>*) you specified in the command line. The .ear file contains the generated EJB, JAR, and XML files for your application, where the *application.xml* file located in the */deploy/META-INF* directory for UNIX or the *\deploy\META-INF* directory for Windows serves as the EAR manifest file.
- An archive JAR file containing your EJB application class files is located within the */deploy* directory for UNIX or the *\deploy* directory for Windows. The JAR file includes all EJB application class files and the deployment descriptor file.
- A standard J2EE EJB deployment descriptor (*ejb-jar.xml*) for the generated bean in the module is located within the */classes/META-INF* directory for UNIX or the *\classes\META-INF* directory for Windows. The XML deployment descriptor describes the application components and provides additional information to enable the container to manage the application.
- The source code of a set of Java classes that you can use in your Java applications is located within the */src* directory for UNIX or the *\src* directory for Windows. The generated JavaBean and EJB Java source code is contained in subdirectories according to their Java package name. An EJB client stub is also generated.
- The */classes* directory for UNIX or the *\classes* directory for Windows contains the compiled generated classes and additional XML resources used by the generated code.

## Advanced Configuration

To have more controls on the EJB generated from a WSDL document, an XML configuration file can be supplied to the *wsdl2ejb* utility. Through the configuration file, developers can control several options on the WSDL source, as well as options on the generated EJB.

Developers can also use the configuration file to supply additional xml to Java type maps, so that WSDL documents using complex types can be supported.

The syntax of the *wsdl2ejb* configuration file is shown in its Document Type Definition (DTD) as follows:

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- Specify the properties of the source WSDL document and of the target EJB. -->
<!ELEMENT wsdl2ejb (useProxy?, useWallet?, wsdl, ejb?, mapTypes?)>

<!-- Specify if the generated EJB should use the supplied HTTP proxy when accessing HTTP URLs -->
<!ELEMENT useProxy (#PCDATA)>
<!ATTLIST useProxy
    proxyHost CDATA #REQUIRED
    proxyPort CDATA #REQUIRED>

<!-- Specify the location of the wallet credential file used by the generated EJB for opening HTTPS connection -->
<!ELEMENT useWallet (#PCDATA)>
<!ATTLIST useWallet
    location CDATA #REQUIRED>

<!--
Specify how the wsdl2ejb tools should process the source WSDL document.
In addition to the mandatory location of the WSDL document, the name of the WSDL service and
its port can be specified. In this case, an EJB will be generated only for the supplied service and
port.
An alternative: the name of a WSDL service binding and the SOAP location to be used can be supplied.
In the latter case, an EJB using the specified binding and the supplied SOAP location will be used.
This is particularly useful when generating an EJB from a WSDL stored in a UDDI registry.
-->
```

In fact, following a UDDI best practice, the WSDL SOAP location will be managed separately from the WSDL document.

```
-->
<!ELEMENT wsdl (location, ((service-name, service-port) | (service-binding, soap-location)))?>

<!-- Specify the location of the source WSDL document (for example, "/home/mywsdl.wsdl",
"http://myhost/mywsdl.wsdl") -->
<!ELEMENT location (#PCDATA)>

<!-- Specify the name of the WSDL service to be used for the generation.
It is the name of one of the services defined in the source WSDL. -->
<!ELEMENT service-name (#PCDATA)>

<!-- Specify the service port of the WSDL service to be used for the generation.
It is the name of one ports of the service name defined above in the source WSDL. -->
<!ELEMENT service-port (#PCDATA)>

<!-- Specify the name of the WSDL binding to be used for the generation.
It is the name of one of the bindings defined in the source WSDL. -->
<!ELEMENT service-binding (#PCDATA)>

<!-- Specify the SOAP location service port of the WSDL service to be used for the generation.
It is the name of one ports of the service name defined above in the source WSDL. -->
<!ELEMENT soap-location (#PCDATA)>

<!-- Specify the properties related to the generated EJB. -->
<!ELEMENT ejb (application-name?, ejb-name?, package-name?, remote-name?, session-type?)>

<!-- Specify the name of the J2EE application for the generated EAR. -->
<!ELEMENT application-name (#PCDATA)>

<!-- Specify the JNDI binding key name for the generated EJB. -->
<!ELEMENT ejb-name (#PCDATA)>

<!-- Specify the name for Java package under which the generated EJB will belong. (for example, com.oracle) -->
<!ELEMENT package-name (#PCDATA)>

<!-- Specify the class name for the EJB Remote Interface (for example, MyWsdlEjb) -->
<!ELEMENT remote-name (#PCDATA)>

<!-- Specify the if the generated EJB should be stateless or stateful (for example, Stateless | Stateful) -->
<!ELEMENT session-type (#PCDATA)>

<!--
Specify the custom Java types and map them to XML types.
The JAR attribute value will point to a JAR file containing the definition of the custom
types or the serializer/deserializer to be used for the custom type.
-->
<!ELEMENT mapTypes (map*)>
<!--ATTLIST mapTypes
jar          CDATA    #IMPLIED>

!--
Specify a new XML to JAR type map.
EncodingStyle: name of the encodingStyle under which this map will belong
                (for example, http://schemas.xmlsoap.org/soap/encoding/)
namespace-uri  : uri of the namespace for the XML type defined in this map
local-name    : localname of the XML type defined in this map
java-type     : Java class name to which this type is mapped to (for example, com.org.MyBean)
java2xml-class-name: Java class name of the type serializer
                (for example, org.apache.soap.encoding.soapenc.BeanSerializer)
xml2java-class-name: Java class name of the type deserializer
                (for example, org.apache.soap.encoding.soapenc.BeanSerializer)
-->
```

```
-->
<!ELEMENT map (#PCDATA)>
<!ATTLIST map
    encodingStyle      CDATA #REQUIRED
    namespace-uri      CDATA #REQUIRED
    local-name         CDATA #REQUIRED
    java-type          CDATA #REQUIRED
    java2xml-class-name CDATA #REQUIRED
    xml2java-class-name CDATA #REQUIRED>
```

**Table 11–2** describes the elements, subelements, and attributes of the `wsdl2ejb` XML configuration file as defined in the DTD. Required elements and attributes are shown as **bold** text.

**Table 11–2** Elements, Subelements, and Attributes of the `wsdl2ejb` XML Configuration File as Defined in the DTD

Element	Subelement	Attribute	Description
useProxy			Optional element. Specifies the proxy server attributes.
		<b>proxyHost</b>	Required attribute. Specifies the host name of the proxy server.
		<b>proxyPort</b>	Required attribute. Specifies the port number of the proxy server.
useWallet			Optional element. Specifies the Oracle Wallet attribute.
		<b>location</b>	Required attribute. Specifies the location of the Oracle Wallet credential file used by the EJB for opening the HTTPS connection.
wsdl			Required element. Specifies how the <code>wsdl2ejb</code> utility should process the source WSDL document. Requires the location element be specified and optionally, either the service-name and service-port pair of elements or the service-binding and soap-location pair of elements be specified.
		<b>location</b>	Required element. Specifies the location of the source WSDL document. Can be a file path or an URL.
		service-name	Optional element. Specifies the name of the WSDL service to be used for the generated EJB. If specified, must be specified with the service-port element as a pair of elements.
		service-port	Optional element. Specifies the service port of the WSDL service to be used for the generated EJB. If specified, must be specified with the service-name element as a pair of elements.
		service-binding	Optional element. Specifies the name of the WSDL binding to be used for the generated EJB. If specified, must be specified with the soap-location element as a pair of elements.
		soap-location	Optional element. Specifies the SOAP location service port of the WSDL service to be used for the generated EJB. If specified, must be specified with the service-binding element as a pair of elements.
ejb			Optional element. Specifies the properties related to the generated EJB.
		application-name	Optional element. Specifies the name of the J2EE application for the generated EAR file.
		ejb-name	Optional element. Specifies the JNDI binding key name for the generated EJB.

**Table 11–2 (Cont.) Elements, Subelements, and Attributes of the wsdl2ejb XML Configuration File as Defined in the DTD**

Element	Subelement	Attribute	Description
	package-name		Optional element. Specifies the name for the Java package under which the generated EJB belongs.
	remote-name		Optional element. Specifies the class name for the EJB Remote Interface.
	session-type		Optional element. Specifies whether the generated EJB should be stateless or stateful.
mapTypes			Optional element. Specifies the custom Java types and maps them to XML types.
	map		Optional element. Specifies the XML to JAR type map.
		<b>encodingStyle</b>	Required attribute. Specifies the name of the encoding style under which this map belongs.
		<b>namespace-uri</b>	Required attribute. Specifies the URI of the namespace for the XML type defined in this map.
		<b>local-name</b>	Required attribute. Specified the local name of the XML type defined in this map.
		<b>java-type</b>	Required attribute. Specifies the Java class name to which this type is mapped.
		<b>java2xml-class-name</b>	Required attribute. Specifies the Java class name of the type serializer.
		<b>xml2java-class-name</b>	Required attribute. Specifies the Java class name of the type deserializer.

Developers can run the `wsdl2ejb` utility with a configuration file using the following command:

```
java -jar wsdl2ejb.jar -conf wsdlconf.xml
```

### Supported WSDL Documents

The `wsdl2ejb` utility supports most WSDL documents using SOAP binding. This support includes both Remote Procedure Call (RPC) and document style documents as well as types that are encoded or literal. [Table 11–3](#) shows how the supported XML Schema types are mapped to the corresponding Java type by default. Any other required type will have to be supported though the custom type mapping described previously.

**Table 11–3 Supported XML Schema Types and Corresponding Java Type**

Supported XML Schema Type	Corresponding Java Type
string	java.lang.String
int	int
decimal	BigDecimal
float	float
double	double
Boolean	Boolean
long	long

**Table 11-3 (Cont.) Supported XML Schema Types and Corresponding Java Type**

Supported XML Schema Type	Corresponding Java Type
short	short
byte	byte
date	GregorianCalendar
timeInstant	java.util.Date

---

**Note:** Arrays of supported types, shown in [Table 11-3](#) are also supported.

---

## Known Limitations of the `wsdl2ejb` Utility

The following information describes the known limitations of the `wsdl2ejb` utility:

- Supports only types defined by the W3C recommendation XML schema version whose namespace is: `http://www.w3.org/2001/XMLSchema`
- Supports only the One-way and Request-Response transmission primitives defined in the WSDL 1.1 specification.
- Does not support WSDL documents that use the `<import>` tag to include other WSDL documents.
- Does not support HTTP, MIME, or any other custom bindings.

## Running the Demonstration

The `wsdl2ejb` demo directory contains examples on how to use the `wsdl2ejb` utility. All the commands are assumed to be executed from the `$ORACLE_HOME/webservices/demo/basic/wsdl2ejb` directory. The demonstration (demo) will use some sample WSDL documents as sources and generate EJB that can be used to invoke the Web Service operations.

The demos can be run using Jakarta ant. Review the `build.xml` file to make sure that the initial properties (`RMI_HOST`, `RMI_PORT`, `RMI_ADMIN`, `RMI_PWD`) are set correctly according to your configuration. The `build.xml` file will execute the `wsdl2ejb` utility on the demo WSDL documents, deploy the generated EJB, and execute the EJB clients.

---

**Note:** If you are executing the demos behind a firewall and need to set proxy information to access external HTTP sites, make sure this proxy information is specified in the `wsdl2ejb` configuration files (`rpc_doc_conf.xml`, `base_conf.xml`).

---

---

---

**Note:** The demos are based on WSDL/SOAP interoperability test suites. They access live SOAP services available on the Internet as SOAP interoperability test cases. The successful execution of these demos depends on the availability of these services.

---

---

The directory structure of the demos is as follows:

demo/web\_services/wsdl2ejb:

- README.txt : Readme file
- build.xml : Jakarta ant build file to run all the demos
- rpc\_doc : directory for simple RPC and document style operations
  - rpc\_doc\_conf.xml : wsdl2ejb configuration file for the rpc\_doc demo
  - TestRpcDocClient.java : client for the rpc\_doc demo
  - DocAndRpc.wsdl : sample WSDL for the rpc\_doc demo
  - (generated) : directory where the EJB will be generated
- base
  - base\_conf.xml : wsdl2ejb configuration file for the base interoperability demo
  - TestInteropBaseClient.java : client for the base interoperability demo
  - InteropTest.wsdl : WSDL document for the base interoperability demo
  - MySoapStructBean.java : bean utilized to map the custom type used in the example defined in the WSDL document
  - MySoapStructBean.jar : packaged-compiled custom type bean
  - (generated) : directory where the EJB will be generated

### RPC and Document Style with Simple Types Example

This example uses a simple WSDL document that shows a couple of operations: Add and Multiply. Add is using the document-style operation using literal parts, while Multiply is RPC-style and uses encoded parts.

To generate the EJB stub, use the following command:

```
On UNIX
cd $ORACLE_HOME/webservices/demo/basic/wsdl2ejb
java -jar ../../lib/wsdl2ejb.jar -conf rpc_doc/rpc_doc_conf.xml
```

```
On Windows
cd %ORACLE_HOME%\webservices\demo\basic\wsdl2ejb
java -jar ..\..\lib\wsdl2ejb.jar -conf rpc_doc\rpc_doc_conf.xml
```

The utility generates the `TestApp.ear` file containing the definition of a stateless EJB, which can be used as a proxy for the Web Service. The EAR file can be deployed in OC4J as any standard EJB. Refer to *Oracle Application Server Containers for J2EE User's Guide* for information on how to deploy an EJB.

By looking at the generated EJB Remote Interface, you can see how the WSDL portType `DocAndRpc.wsdl` file has been mapped to Java.

WSDL PortType:

```
<types>
  <s:schema elementFormDefault="qualified" targetNamespace="http://soapinterop.org">
    <s:element name="Add">
      <s:complexType>
        <s:sequence>
          <s:element minOccurs="1" maxOccurs="1" name="a" type="s:int" />
          <s:element minOccurs="1" maxOccurs="1" name="b" type="s:int" />
        </s:sequence>
      </s:complexType>
    </s:element>
  </s:schema>
</types>
```

```

        </s:complexType>
    </s:element>
    <s:element name="AddResponse">
        <s:complexType>
            <s:sequence>
                <s:element minOccurs="1" maxOccurs="1" name="AddResult" type="s:int" />
            </s:sequence>
        </s:complexType>
    </s:element>
</s:schema>
</types>
<message name="AddSoapIn">
    <part name="parameters" element="s0:Add" />
</message>
<message name="AddSoapOut">
    <part name="parameters" element="s0:AddResponse" />
</message>
<message name="MultiplySoapIn">
    <part name="a" type="xsd:int" />
    <part name="b" type="xsd:int" />
</message>
<message name="MultiplySoapOut">
    <part name="MultiplyResult" type="s:int" />
</message>
<portType name="TestSoap">
    <operation name="Add">
        <input message="s0:AddSoapIn" />
        <output message="s0:AddSoapOut" />
    </operation>
    <operation name="Multiply">
        <input message="s0:MultiplySoapIn" />
        <output message="s0:MultiplySoapOut" />
    </operation>
</portType>

```

From the `Test.java` file, the EJB Remote Interface is:

```

public org.w3c.dom.Element add(org.w3c.dom.Element parameters)
    throws RemoteException;

public int multiply(int a, int b)
    throws RemoteException;

```

When the WSDL operation is using RPC style and its parts are encoded, the parts XML schema type is mapped to a corresponding Java native type. In this example, `xsd:int` is mapped to Java `int`. In a document style using literal parts, each part is simply mapped to an `org.w3c.dom.Element`.

The following client code in the `TestRpcDocClient.java` file can be used to invoke the Add and Multiply Web Service operations. The code has been produced by modifying the client code stub generated by the `wsdl2ejb` utility.

```

import java.io.*;
import java.util.*;
import javax.naming.*;

import org.w3c.dom.*;
import oracle.xml.parser.v2.*;

import org.mssoapinterop.asmx.Test;
import org.mssoapinterop.asmx.TestHome;

/**
 * This is a simple client template. To compile it,

```

```
* please include the generated EJB jar file as well as
* EJB and JNDI libraries in classpath.
*/
public class TestRpcDocClient
{
    // replace the values
    private static String RMI_HOST = "localhost";
    private static String RMI_PORT = "23791";
    private static String RMI_ADMIN = "admin";
    private static String RMI_PWD = "welcome";

    public TestRpcDocClient () {}

    public static void main(String args[]) {

        TestRpcDocClient client = new TestRpcDocClient();

        try {

            RMI_HOST = args[0];
            RMI_PORT = args[1];
            RMI_ADMIN = args[2];
            RMI_PWD = args[3];

            Hashtable env = new Hashtable();
            env.put(Context.INITIAL_CONTEXT_FACTORY, "com.evermind.server.rmi.RMIInitialContextFactory");
            env.put(Context.SECURITY_PRINCIPAL, RMI_ADMIN);
            env.put(Context.SECURITY_CREDENTIALS, RMI_PWD);
            env.put(Context.PROVIDER_URL, "ormi://" + RMI_HOST + ":" + RMI_PORT + "/Wsd12EjbTestApp1");
            Context ctx = new InitialContext(env);
            TestHome home = (TestHome) ctx.lookup("msssoapinterop.org/asmx/DocAndRpc.asmx");

            Test service = home.create();

            // call any of the Remote methods that follow to access the EJB

            //
            // Add test
            //
            Document doc = new XMLDocument();
            Element elAdd = doc.createElementNS("http://soapinterop.org", "s:Add");
            Element elA = doc.createElementNS("http://soapinterop.org", "s:a");
            Element elB = doc.createElementNS("http://soapinterop.org", "s:b");
            elA.appendChild(doc.createTextNode("4"));
            elB.appendChild(doc.createTextNode("3"));
            elAdd.appendChild(elA);
            elAdd.appendChild(elB);
            doc.appendChild(elAdd);

            Element elAddResponse = service.add(elAdd);
            Node tNode = elAddResponse.getFirstChild().getFirstChild();
            System.out.println("AddResponse: " + tNode.getNodeValue());

            //
            // Multiply Test
            //
            int a = 4;
            int b = 3;
            int iMultiplyResponse = service.multiply(a, b);
            System.out.println("MultiplyResponse: " + iMultiplyResponse);

        }
        catch (Throwable ex) {
            ex.printStackTrace();
        }
    }
}
```



```
}
}
```

The result of the execution of the client is the following:

```
AddResponse: 7
MultiplyResponse: 12
```

## Round 2 Interop Services: Base Test Suite Example

This example starts from a subset of the WSDL document defined by the base test suite of the second round of SOAP interoperability tests. The purpose of this demo example is to show the usage of built-in types in the SOAP Mapping Registry as well as how to add custom types mapping.

Start by looking at the WSDL portType in the `InteropTest.wsdl` file.

```
<types>
  <schema xmlns="http://www.w3.org/2001/XMLSchema"
    targetNamespace="http://soapinterop.org/xsd">
    <complexType name="ArrayOfstring">
      <complexContent>
        <restriction base="SOAP-ENC:Array">
          <attribute ref="SOAP-ENC:arrayType" wsdl:arrayType="string[]"/>
        </restriction>
      </complexContent>
    </complexType>
    <complexType name="ArrayOfint">
      <complexContent>
        <restriction base="SOAP-ENC:Array">
          <attribute ref="SOAP-ENC:arrayType" wsdl:arrayType="int[]"/>
        </restriction>
      </complexContent>
    </complexType>
    <complexType name="ArrayOffloat">
      <complexContent>
        <restriction base="SOAP-ENC:Array">
          <attribute ref="SOAP-ENC:arrayType" wsdl:arrayType="float[]"/>
        </restriction>
      </complexContent>
    </complexType>
    <complexType name="ArrayOfSOAPStruct">
      <complexContent>
        <restriction base="SOAP-ENC:Array">
          <attribute ref="SOAP-ENC:arrayType"
            wsdl:arrayType="s:SOAPStruct[]"/>
        </restriction>
      </complexContent>
    </complexType>
    <complexType name="SOAPStruct">
      <all>
        <element name="varString" type="string"/>
        <element name="varInt" type="int"/>
        <element name="varFloat" type="float"/>
      </all>
    </complexType>
  </schema>
</types>

<message name="echoStringRequest">
```

```
    <part name="inputString" type="xsd:string"/>
</message>
<message name="echoStringResponse">
    <part name="return" type="xsd:string"/>
</message>
<message name="echoStringArrayRequest">
    <part name="inputStringArray" type="s:ArrayOfstring"/>
</message>
<message name="echoStringArrayResponse">
    <part name="return" type="s:ArrayOfstring"/>
</message>
<message name="echoIntegerRequest">
    <part name="inputInteger" type="xsd:int"/>
</message>
<message name="echoIntegerResponse">
    <part name="return" type="xsd:int"/>
</message>
<message name="echoIntegerArrayRequest">
    <part name="inputIntegerArray" type="s:ArrayOfint"/>
</message>
<message name="echoIntegerArrayResponse">
    <part name="return" type="s:ArrayOfint"/>
</message>
<message name="echoFloatRequest">
    <part name="inputFloat" type="xsd:float"/>
</message>
<message name="echoFloatResponse">
    <part name="return" type="xsd:float"/>
</message>
<message name="echoFloatArrayRequest">
    <part name="inputFloatArray" type="s:ArrayOffloat"/>
</message>
<message name="echoFloatArrayResponse">
    <part name="return" type="s:ArrayOffloat"/>
</message>
<message name="echoStructRequest">
    <part name="inputStruct" type="s:SOAPStruct"/>
</message>
<message name="echoStructResponse">
    <part name="return" type="s:SOAPStruct"/>
</message>
<message name="echoStructArrayRequest">
    <part name="inputStructArray" type="s:ArrayOfSOAPStruct"/>
</message>
<message name="echoStructArrayResponse">
    <part name="return" type="s:ArrayOfSOAPStruct"/>
</message>
<message name="echoVoidRequest"/>
<message name="echoVoidResponse"/>
<message name="echoBase64Request">
    <part name="inputBase64" type="xsd:base64Binary"/>
</message>
<message name="echoBase64Response">
    <part name="return" type="xsd:base64Binary"/>
</message>
<message name="echoDateRequest">
    <part name="inputDate" type="xsd:dateTime"/>
</message>
<message name="echoDateResponse">
    <part name="return" type="xsd:dateTime"/>
```

```

</message>
<message name="echoDecimalRequest">
  <part name="inputDecimal" type="xsd:decimal" />
</message>
<message name="echoDecimalResponse">
  <part name="return" type="xsd:decimal" />
</message>
<message name="echoBooleanRequest">
  <part name="inputBoolean" type="xsd:boolean" />
</message>
<message name="echoBooleanResponse">
  <part name="return" type="xsd:boolean" />
</message>

<portType name="InteropTestPortType">
  <operation name="echoString" parameterOrder="inputString">
    <input message="tns:echoStringRequest" />
    <output message="tns:echoStringResponse" />
  </operation>
  <operation name="echoStringArray" parameterOrder="inputStringArray">
    <input message="tns:echoStringArrayRequest" />
    <output message="tns:echoStringArrayResponse" />
  </operation>
  <operation name="echoInteger" parameterOrder="inputInteger">
    <input message="tns:echoIntegerRequest" />
    <output message="tns:echoIntegerResponse" />
  </operation>
  <operation name="echoIntegerArray" parameterOrder="inputIntegerArray">
    <input message="tns:echoIntegerArrayRequest" />
    <output message="tns:echoIntegerArrayResponse" />
  </operation>
  <operation name="echoFloat" parameterOrder="inputFloat">
    <input message="tns:echoFloatRequest" />
    <output message="tns:echoFloatResponse" />
  </operation>
  <operation name="echoFloatArray" parameterOrder="inputFloatArray">
    <input message="tns:echoFloatArrayRequest" />
    <output message="tns:echoFloatArrayResponse" />
  </operation>
  <operation name="echoStruct" parameterOrder="inputStruct">
    <input message="tns:echoStructRequest" />
    <output message="tns:echoStructResponse" />
  </operation>
  <operation name="echoStructArray" parameterOrder="inputStructArray">
    <input message="tns:echoStructArrayRequest" />
    <output message="tns:echoStructArrayResponse" />
  </operation>
  <operation name="echoVoid">
    <input message="tns:echoVoidRequest" />
    <output message="tns:echoVoidResponse" />
  </operation>
  <operation name="echoBase64" parameterOrder="inputBase64">
    <input message="tns:echoBase64Request" />
    <output message="tns:echoBase64Response" />
  </operation>
  <operation name="echoDate" parameterOrder="inputDate">
    <input message="tns:echoDateRequest" />
    <output message="tns:echoDateResponse" />
  </operation>
  <operation name="echoDecimal" parameterOrder="inputDecimal">

```

```
        <input message="tns:echoDecimalRequest" />
        <output message="tns:echoDecimalResponse" />
    </operation>
    <operation name="echoBoolean" parameterOrder="inputBoolean">
        <input message="tns:echoBooleanRequest" />
        <output message="tns:echoBooleanResponse" />
    </operation>
</portType>
```

Notice that the WSDL document contains more complex types than the previous demo. Array of primitives types are now used as well as the struct primitive types. With the exception of the SOAPStruct complex type, every other type is supported as built-in type in the SOAP Mapping Registry. You then need to add a new complex type definition to the SOAP Mapping Registry to handle the SOAPStruct complex type.

The SOAPStruct schema definition is the following:

```
<complexType name="SOAPStruct">
    <all>
        <element name="varString" type="string" />
        <element name="varInt" type="int" />
        <element name="varFloat" type="float" />
    </all>
</complexType>
```

In the `MySoapStructBean.java` file, this SOAPStruct complex type can be mapped to a simple JavaBean class such as the following, and have the marshalling and unmarshalling actions handled by the `BeanSerializer`.

```
public class MySoapStructBean implements java.io.Serializable
{
    private String m_varString = null;
    private int m_varInt = 0;
    private float m_varFloat = 0;

    public MySoapStructBean() {}
    public MySoapStructBean(String s, int i, float f) {
        m_varString = s;
        m_varInt = i;
        m_varFloat = f;
    }

    public String getVarString () { return m_varString; }
    public int getVarInt() { return m_varInt; }
    public float getVarFloat() { return m_varFloat; }

    public void setVarString (String s) { m_varString = s; }
    public void setVarInt(int i) { m_varInt = i; }
    public void setVarFloat(float f) { m_varFloat = f; }
}
```

With the mapping JavaBean class ready, and having identified what serializer and deserializer to use, you can now configure the `wsdl2ejb` utility so that a new schema to Java map is added. This can be achieved by adding the following to the `wsdl2ejb` configuration file, `base_conf.xml`:

```
<mapTypes jar="base/MySoapStructBean.jar" >
    <map encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
        local-name="SOAPStruct"
        namespace-uri="http://soapinterop.org/xsd"
        java-type="MySoapStructBean"
```

```

        java2xml-class-name="org.apache.soap.encoding.soapenc.BeanSerializer"
        xml2java-class-name="org.apache.soap.encoding.soapenc.BeanSerializer" />
</mapTypes>

```

The `MySoapStructBean.jar` file contains the definition of the `MySoapStructBean` class. With this map, the SOAPStruct complex type, belonging to the `http://soapinterop.org/xsd` namespace, will be mapped to the `MySoapStructBean` JavaBean class and the converse is true as well. For more information about SOAP serializers and deserializers, see the Oracle SOAP documentation.

With this additional configuration, you can now run the `wSDL2ejb` utility with the following command:

```

On UNIX
cd $ORACLE_HOME/webservices/demo/basic/wSDL2ejb
java -jar ../../../../lib/wSDL2ejb.jar -conf base/base_conf.xml
On Windows
cd %ORACLE_HOME%\webservices/demo/basic/wSDL2ejb
java -jar ../../../../lib\wSDL2ejb.jar -conf base\base_conf.xml

```

The `wSDL2ejb` utility generates the `InteropLabApp.ear` file that contains the definition of a stateless EJB, which can be used as a proxy for the Web Service. The EAR file can be deployed in OC4J as any standard EJB. See *Oracle Application Server Containers for J2EE User's Guide* for information on how to deploy an EJB.

The `TestInteropBaseClient.java` class file, saved in the base directory, can be used to test the generated EJB after it has been deployed. The result of the execution of the client is the following:

```

echoString: Hello World!
echoStringArray[0]: Hello World!
echoStringArray[1]: Seems to work!
echoStringArray[2]: Fine!
echoStringArray[3]: WOW
echoInteger: 7
echoIntegerArray[0]: 1
echoIntegerArray[1]: 2
echoIntegerArray[2]: 3
echoIntegerArray[3]: 4
echoFloat: 1.7777
echoFloatArray[0]: 1.1
echoFloatArray[1]: 1.2
echoFloatArray[2]: 1.3
echoFloatArray[3]: 1.4
echoStruct: varString=Hello World , varInt=1 , varFloat=1.777
echoStructArray: varString[0]=Hello World , varInt[0]=0 , varFloat=[0]=1.7771
echoStructArray: varString[1]=Hello World 1 , varInt[1]=1 , varFloat=[1]=1.7772
echoStructArray: varString[2]=Hello World 2 , varInt[2]=2 , varFloat=[2]=1.7773
echoStructArray: varString[3]=Hello World 3 , varInt[3]=3 , varFloat=[3]=1.7774
echoVoid.
echoDecimal: 1.77709999999999990194510246510617434978485107421875
echoBoolean: true
echoBase64[0]: 1
echoBase64[1]: 2
echoBase64[2]: 3
echoBase64[3]: 4
echoDate: Sat Nov 10 12:30:00 EST 2001

```

## Dynamic Invocation of Web Services

When a Java2 Platform Enterprise Edition (J2EE) application acquires a WSDL document at runtime, the dynamic invocation API is used to invoke any SOAP operation described in the WSDL document. The dynamic invocation API describes a `WebServiceProxyFactory` factory class that can be used to build instances of a `WebServiceProxy`. Each created `WebServiceProxy` instance is based on the location of the WSDL document, (and optionally on additional qualifiers), that identify which service and port should be used. The `WebServiceProxy` class exposes methods to determine the WSDL `portType`, including the syntax and signatures of all operations exposed by the WSDL service and to invoke the defined operations.

This section briefly describes the dynamic invocation API and how to use it.

For Java samples, refer to the code supplied with Oracle Application Server Web Services in `$ORACLE_HOME/webservices/demo/basic/java_services/dynamicproxy` on UNIX or in `%ORACLE_HOME%\webservices\demo\basic\java_services\dynamicproxy` on Windows. For EJB samples, refer to the code supplied in the directory `$ORACLE_HOME/webservices/demo/basic/stateless_ejb` on UNIX or `%ORACLE_HOME%\webservices\demo\basic\stateless_ejb` on Windows.

### Dynamic Invocation API

The dynamic invocation API contains two packages, `oracle.j2ee.ws.client` and `oracle.j2ee.ws.client.wsdl`, which contain additional classes grouped by interface, class, and exception, as shown in [Table 11-4](#) and [Table 11-5](#).

**Table 11-4** The `oracle.j2ee.ws.client` Package

Classes	Description
<b>Classes</b>	
<code>WebServiceProxyFactory</code>	This class creates a <code>WebServiceProxy</code> class given a WSDL document.
<b>Interfaces</b>	
<code>WebServiceProxy</code>	This interface represents a service defined in a WSDL document.
<code>WebServiceMethod</code>	This interface invokes a Web Service method.
<b>Exceptions</b>	
<code>WebServiceProxyException</code>	This class describes exceptions raised by the <code>WebServiceProxy</code> API.

**Table 11-5** The `oracle.j2ee.ws.client.wsdl` Package

Classes	Description
<b>Interfaces</b>	
<code>PortType</code>	This interface represents a port type.
<code>Operation</code>	This interface represents a WSDL operation.
<code>Input</code>	This interface represents an input message, and contains the name of the input and the message itself.
<code>Output</code>	This interface represents an output message, and contains the name of the output and the message itself.

**Table 11–5 (Cont.) The oracle.j2ee.ws.client.wsdl Package**

Classes	Description
Fault	This interface represents a fault message, and contains the name of the fault and the message itself.
Message	This interface describes a message used for communication with an operation.
Part	This interface represents a message part and contains the part's name, elementName, and typeName.
<b>Classes</b>	
OperationType	This class represents an operation type which can be one of request-response, solicit response, one way, or notification.

The oracle.j2ee.ws.client package is described in more detail in this section. The API documentation describes to use this proxy API can be found in the Oracle Application Server 10g Documentation Library as Proxy API Reference (Javadoc) under Oracle Application Server Web Services, which is located under the J2EE and Internet Applications tab.

The WebServiceProxyFactory class contains methods that can instantiate a WebServiceProxy class given either the URL or the Java input stream of the WSDL document. Four methods let you use either the first service and its first port in the supplied WSDL document or use the name of one of services and the name of one of the ports of the service to create a WebServiceProxy instance. Two methods also let you create a WebServiceProxy instance for a WSDL document, which has been authored following the UDDI best practices for WSDL. A method lets you supply additional optional initialization parameters to the WebServiceProxy instance.

Table 11–6 briefly describes the WebServiceProxyFactory factory class methods and the required parameters for each method. See the JavaDoc for more detailed information about this factory class and its methods.

**Table 11–6 WebServiceProxyFactory Factory Methods and Parameters**

Methods	Parameters
createWebServiceProxy()	java.io.InputStream isWsdL java.net.URL baseUrl
createWebServiceProxy()	java.net.URL wsdlURL
createWebServiceProxyFromBinding()	java.io.InputStream wsdlis java.net.URL baseUrl java.lang.String szBindingName java.lang.String szSoapLocation
createWebServiceProxyFromService()	java.io.InputStream wsdlis java.net.URL baseUrl java.lang.String szServiceName java.lang.String szServicePort
createWebServiceProxyFromBinding()	java.net.URL wsdlUrl java.lang.String szBindingName java.lang.String szSoapLocation
createWebServiceProxyFromService()	java.net.URL wsdlUrl java.lang.String szServiceName java.lang.String szServicePort
setProperty()	java.util.Hashtable ht

[Table 11-7](#) describes the `WebServiceProxy` interface. The `WebServiceProxyFactory` factory methods optionally take additional parameters that are provided in the `WebServiceProxy` interface that can be used to dynamically invoke an operation in a WSDL document.

**Table 11-7** *WebServiceProxy Interface Methods and Parameters*

Methods	Parameters	Description
<code>getXMLMappingRegistry()</code>	None	Returns the SOAP mapping registry used by the <code>WebServiceProxy</code> and contains information that lets clients use this registry to query for XML to or from Java type mapping as well as extend the mapping registry with new map definitions.
<code>getPortType()</code>	None	Returns a structure describing the WSDL portType used by this proxy and contains information about operations associated with this port type.
<code>getMethod()</code>		Returns a <code>WebServiceMethod</code> method, which can be used to invoke Web Service methods.
	<code>szOperationName</code> <code>szInputName</code> <code>szOutputName</code>	Name of the WSDL operation to be executed. Name of the <code>wsdl:input</code> tag for the operation to be executed. Name of the <code>wsdl:output</code> tag for the operation to be executed.
<code>getMethod()</code>		Returns a <code>WebServiceMethod</code> method, which can be used to invoke Web service methods and provides a signature that can be used for non-overloaded WSDL operations.
	<code>szOperationName</code>	Name of the WSDL operation to be executed.

[Table 11-8](#) describes the `WebServiceMethod` interface, which is used to invoke a Web Service method.

**Table 11-8** *WebServiceMethod Interface Methods and Parameters*

Methods	Parameters	Description
<code>getInputEncodingStyle()</code>	None	Returns the encoding style to be used by the input message parts, null if none has been specified in the source WSDL.
<code>getOutputEncodingStyle()</code>	None	Returns the encoding style to be used by the output message parts, null if none has been specified in the source WSDL.
<code>invoke()</code>		Executes one of the service operations with the set of supplied input parts and returns the object, if the response message contains only one part, return the response part, otherwise an array of the output message parts. If the invoked WSDL operation has no output parts, null will be returned.
	<code>inMsgPartNames</code> <code>inMsgPartValues</code>	Name of the parts supplied in the input message. Corresponding value of the parts whose name is supplied in the <code>inMsgPartNames</code> parameter. If the invoked WSDL operation has no input parts, null or empty arrays parameters can be supplied

The `oracle.j2ee.ws.client.wsdl` package exposes methods to determine the WSDL portType, including the syntax and signatures of all operations exposed by the WSDL service.



## WebServiceProxy Client

The following client code shows the use of the dynamic invocation API followed by the output of the client execution. The client code shows the following:

- Initializes proxy parameters in the `WebServiceProxyFactory`.
- Creates an instance of the proxy given a URL of a WSDL document.
- Performs WSDL introspection.
- Shows the input message parts.
- Executes a Web Service operation with a set of supplied input parts and returns the result.

The WSDL document is described as follows:

```
<?xml version="1.0" encoding="utf-8" ?>
- <definitions xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/" xmlns:tns="http://soapinterop.org"
xmlns:s="http://www.w3.org/2001/XMLSchema" xmlns:http="http://schemas.xmlsoap.org/wsdl/http/"
xmlns:tm="http://microsoft.com/wsdl/mime/textMatching/" xmlns:mime="http://schemas.xmlsoap.org/wsdl/mime/"
xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/" targetNamespace="http://soapinterop.org"
xmlns="http://schemas.xmlsoap.org/wsdl/">
  <types />
  - <message name="AddSoapIn">
    <part name="a" type="s:int" />
    <part name="b" type="s:int" />
  </message>
  - <message name="AddSoapOut">
    <part name="AddResult" type="s:int" />
  </message>
  - <portType name="TestSoap">
    - <operation name="Add">
      <input message="tns:AddSoapIn" />
      <output message="tns:AddSoapOut" />
    </operation>
  </portType>
  - <binding name="TestSoap" type="tns:TestSoap">
    <soap:binding transport="http://schemas.xmlsoap.org/soap/http" style="rpc" />
    - <operation name="Add">
      <soap:operation soapAction="http://soapinterop.org/Add" style="rpc" />
      - <input>
        <soap:body use="encoded" namespace="http://soapinterop.org"
encodingStyle="http://schemas.xmlsoap.org/soap/encoding/" />
      </input>
      - <output>
        <soap:body use="encoded" namespace="http://soapinterop.org"
encodingStyle="http://schemas.xmlsoap.org/soap/encoding/" />
      </output>
    </operation>
  </binding>
  - <service name="Test">
    - <port name="TestSoap" binding="tns:TestSoap">
      <soap:address location="http://mssoapinterop.org/asmx/Rpc.asmx" />
    </port>
  </service>
</definitions>
```

```
package oracle.j2ee.ws.client.impl;

import java.util.*;
import java.io.*;
import java.net.*;
import oracle.j2ee.ws.client.*;
import oracle.j2ee.ws.client.wsdl.*;
import org.apache.soap.util.xml.QName;
```

```
import org.apache.soap.util.xml.XMLJavaMappingRegistry;

public class Client {

    public static void main(String[] args) throws Exception {

        String szWsdUrl = "http://mssoapinterop.org/asmx/Rpc.asmx?WSDL";

        URL urlWsdL = new URL(szWsdUrl);
        System.err.println("WsdL url = " + urlWsdL);

        WebServiceProxyFactory wsfact= new WebServiceProxyFactory();

        //
        // Set some initial parameters
        //
        Hashtable ht = new Hashtable();
        ht.put("http.proxyHost", "www-proxy.us.oracle.com");
        ht.put("http.proxyPort", "80");
        wsfact.setProperties(ht);

        //
        // Create an instance of the proxy
        //
        WebServiceProxy wsp = wsfact.createWebServiceProxy(urlWsdL);

        //
        // Optional: WsdL Introspection
        //
        PortType pt = wsp.getPortType();
        List opList = pt.getOperations();
        for (int i = 0; i < opList.size(); i++) {

            Operation op = (Operation) opList.get(i);
            String szOpName = op.getName();
            String szInput = op.getInput().getName();
            String szOutput = op.getOutput().getName();

            System.err.println("operation["+i+"] = [" + szOpName +
                ", " + szInput + ", " + szOutput + "]);

            //
            // show input message parts
            //
            Message msgIn = op.getInput().getMessage();
            Map mapParts = msgIn.getParts();
            Collection colParts = mapParts.values();
            Iterator itParts = colParts.iterator();

            WebServiceMethod wsm = wsp.getMethod(szOpName);
            String szInEncStyle = wsm.getInputEncodingStyle();
            XMLJavaMappingRegistry xmr = wsp.getXMLMappingRegistry();

            while (itParts.hasNext()) {
                Part part = (Part) itParts.next();
                String szPartName = part.getName();
                QName qname = part.getTypeName();
                String szJavaType = xmr.queryJavaType(qname,
                    szInEncStyle).getName();
                System.err.println("part name = " + szPartName +
```

```

        ", type = " + QName +
        ", java type = " + szJavaType);
    }
}

//
// invoke operation/method Add(2,10)
//
String[] inMsgPartNames = new String[2];
inMsgPartNames[0] = "a";
inMsgPartNames[1] = "b";
Object[] inMsgPartValues = new Object[2];
inMsgPartValues[0] = new Integer(2);
inMsgPartValues[1] = new Integer(10);

WebServiceMethod wsm = wsp.getMethod("Add");
Object objRet = wsm.invoke(inMsgPartNames,
    inMsgPartValues);

System.err.println("Calling method Add(" + inMsgPartValues[0] + ", "
+
    inMsgPartValues[1] + ")");
System.err.println("return = " + objRet);
}
}

```

The output of the client execution is as follows:

```

WsdL url = http://mssoapinterop.org/asmx/Rpc.asmx?WSDL
operation[0] = [Add,,]
part name = b, type = http://www.w3.org/2001/XMLSchema:int, java type = int
part name = a, type = http://www.w3.org/2001/XMLSchema:int, java type = int
Calling method Add(2,10)
return = 12

```

## Known Limitations

The following information describes the known limitations of the dynamic invocation API:

- Supports invoking operations defined in the WSDL document defined by the W3C recommendation XML schema version whose namespace is:  
http://www.w3.org/2001/XMLSchema
- Does not support WSDL documents that use the <import> tag to include other WSDL documents.
- Does not support HTTP, MIME, or any other custom bindings.



---

---

## Advanced Topics for Web Services

This chapter covers advanced Oracle Application Server Web Services topics, including the following topics:

- [Setting the Web Services Debugging Property `ws.debug`](#)
- [Untyped Request Handling Options](#)
- [SOAP Header Support](#)

### Setting the Web Services Debugging Property `ws.debug`

To obtain Oracle Application Server Web Services debugging information, use the Java property `ws.debug`, and set its value to `true`. To set the `ws.debug` value to `true`, use Oracle Enterprise Manager to specify OC4J startup options. Debugging output is sent to the OC4J instance log file corresponding to the island where Oracle Application Server Web Services is running.

[Example 12-1](#) provides sample debugging output.

#### **Example 12-1** Web Services Debug Output

```
WS Debug: initQnameMap('null')
WS Debug: operation name is: helloWorld
WS Debug: QueryString is: invoke=helloWorld&param0=test
WS Debug: Operation Name is: helloWorld
WS Debug: Port Type Local name is: StatelessExamplePortType
WS Debug: Port Type Namespace URI is: http://oracle.j2ee.ws_
example/StatelessExample.wsdl
WS Debug: Operation Local name is: helloWorld
WS Debug: Operation Namespace URI is: http://oracle.j2ee.ws_
example/StatelessExample.wsdl
WS Debug: Operation Get parameter order: null
```

**See Also:** *Oracle Application Server Containers for J2EE User's Guide* for information on setting debugging options and showing debugging output.

### Untyped Request Handling Options

Oracle Application Server Web Services supports requests for RPC style Web Services in the following cases:

- Typed requests where an incoming RPC request with SOAP encoded parameters includes type attributes that specify type information for every incoming parameter. [Example 12-2](#) shows a sample typed RPC request.
- Untyped requests where an incoming RPC request with SOAP encoded parameters may not include a type attribute for every incoming parameter. [Example 12-3](#) shows a sample un-typed RPC request. This type of RPC request provides improved interoperability with .NET clients.

Oracle Application Server Web Services client-side applications and tools do not generate untyped requests, but some external tools or applications may generate such requests. Due to the performance cost for supporting untyped requests, by default such support is not enabled.

To support requests with untyped parameters, use the optional `<accept-untyped-request>` tag with the `WebServicesAssembler`. This tag applies as a sub-tag with the `<stateful-java-service>` and `<stateless-java-service>` tags when the corresponding `<message-style>` tag is set to the value `RPC`. The `<accept-untyped-request>` tag also applies as a sub-tag for the `<stateless-session-ejb-service>` tag.

[Table 12-1](#) shows `<accept-untyped-request>` tag specification.

#### **Example 12-2 Sample Typed RPC Request**

```
<?xml version='1.0' encoding='UTF-8'?>
<SOAP-ENV:Envelope xmlns:SOAP-ENV="
  http://schemas.xmlsoap.org/soap/envelope/"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <SOAP-ENV:Body>
    <ns1:sayHello xmlns:ns1="urn:Hello"
      SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
      <param0 xsi:type="xsd:string">Scott</param0>
      <param1 xsi:type="xsd:int">27</param1>
    </ns1:sayHello>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

#### **Example 12-3 Sample Un-Typed RPC Request**

```
<?xml version='1.0' encoding='UTF-8'?>
<SOAP-ENV:Envelope xmlns:SOAP-ENV="
  http://schemas.xmlsoap.org/soap/envelope/"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <SOAP-ENV:Body>
    <ns1:sayHello xmlns:ns1="urn:Hello"
      SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
      <param0>Scott</param0>
      <param1>27</param1>
    </ns1:sayHello>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

**Table 12–1** *WebServicesAssembler* <accept-untyped-request> Tag

Tag	Description
<accept-untyped-request> value </accept-untyped-request>	Setting <i>value</i> to true tells <i>WebServicesAssembler</i> to allow the Web Service to accept untyped requests. When the value is false, the Web Service does not accept untyped-request.  Valid values: true, false (case is not significant; TRUE and FALSE are also valid)  Default value: false

## SOAP Header Support

This section covers Oracle Application Server Web Services support for SOAP request headers sent from a Web Services client to an endpoint. This section covers the following topics:

- [Client Side SOAP Request Header Support](#)
- [Server Side SOAP Request Header Support](#)
- [Limitations for SOAP Header Support](#)

### Client Side SOAP Request Header Support

Oracle Application Server Web Services generated client-side proxy code provides methods to use SOAP request headers. A SOAP request message, including the SOAP request headers is transmitted to a service endpoint when Web Services proxy code is invoked.

When Oracle Application Server Web Services generates a proxy, either from WSDL for a Web Services Document or RPC style service, the proxy code provides two SOAP request header support methods:

```
void _setSOAPRequestHeaders(org.apache.soap.Header headers)
org.apache.soap.Header _getSOAPRequestHeaders()
```

These methods provide access to an `org.apache.soap.Header` object. By default the `org.apache.soap.Header` object's value is set to null which signifies there are no headers in the SOAP request message. When a request header is needed, use the `_setSOAPRequestHeaders()` method to specify the `Header` object to be sent with the SOAP request message.

---

**Note:** When proxies are generated for Stored Procedure or JMS Document Style Web Services the `_setSOAPRequestHeaders()` and `_getSOAPRequestHeaders()` methods are not supplied.

---

The SOAP request header information is shared for all proxy operations. After the headers are set using `_setSOAPRequestHeaders()`, all subsequent operation invocations using the proxy use the same header value. To set a new header value, call `_setSOAPRequestHeaders()` using a new `Header` object or with a null value.

---

**Note:** After setting the SOAP request header, the same header object is used for each subsequent operation invocation until the object is reset using `_setSOAPRequestHeaders()`.

---

To create and manipulate SOAP request headers you need to populate the header object. The `org.apache.soap.Header` object provides a method for specifying the contents of one or more SOAP header blocks. It is defined as:

```
public void setHeaderEntries(java.util.Vector headerEntries)
```

The vector is populated with `org.w3c.dom.Element` objects which specify individual SOAP header blocks.

When a header entry includes the `mustUnderstand` attribute set to the value 1, the recipient must process the header entry. If the recipient cannot process the header entry, then a SOAP fault is returned with the value `FAULT_CODE_MUST_UNDERSTAND`.

**See Also:** Section 4.2, "SOAP Header", in for information on header entries in SOAP 1.1 <http://www.w3.org/TR/SOAP/>.

### Setting SOAP Headers in a Client-Side Proxy

This section shows a sample that uses the proxy class `EmployeeProxy`. The complete sample containing this code is available in the directory `$ORACLE_HOME/web_services/demo/header_demo/client`. In the sample, a single header block is added to the Header object. The Header object is then supplied as an argument to the proxy's `_setSOAPRequestHeaders()` method.

#### *Example 12-4 Segment of Client Using Message with SOAP Request Header*

```
.  
. .  
// Create an instance of the proxy  
EmployeeProxy proxy = new EmployeeProxy();  
// Create a Header object  
Vector v = new Vector();  
v.add (e);  
Header hdr = new Header();  
hdr.setHeaderEntries(v);  
  
// Set the Header  
proxy._setSOAPRequestHeaders(hdr);  
// Invoke the request  
System.out.println("Salary of MILLER is: " + proxy.getEmployeeSalary("MILLER"));
```

## Server Side SOAP Request Header Support

To process a SOAP request header on the server side, a Web Service needs to implement the `oracle.j2ee.ws.HeaderCallback` interface that is part of the Oracle Application Server Web Services supplied `wsserver.jar`. This interface includes one method that takes a single `org.apache.soap.Header` argument.

The Oracle Application Server Web Services infrastructure calls the `processHeaders()` method before every associated service method.

When an incoming SOAP request header includes one or more header entries with the `mustUnderstand` attribute set to either 1, `true`, or `TRUE` values, then the Web Service implementation must implement the `oracle.j2ee.ws.HeaderCallback` interface. If this interface is not implemented, Oracle Application Server Web Services throws a SOAP fault with the fault code set to `FAULT_CODE_MUST_UNDERSTAND`.



If a Web Service implementation implements the `HeaderCallback` interface, the implementation can throw a SOAP exception with the fault code set to `FAULT_CODE_MUST_UNDERSTAND` if the service does not know how to process a header entry with the `mustUnderstand` attribute set to 1, `true`, or `TRUE`. Oracle Application Server Web Services then translates the exception and Oracle Application Server Web Services throws a SOAP fault with the fault code set to `FAULT_CODE_MUST_UNDERSTAND`.

This section shows server-side Web Services code that provides the implementation for the `Employee` service. The complete sample containing this Web Service is available in the directory `$ORACLE_HOME/web_services/demo/basic/header_demo/client` (after unzipping `$ORACLE_HOME/webservices/demo/demo.zip`).

[Example 12-5](#) shows an interface that extends `HeaderCallback`.

[Example 12-6](#) shows a section of the service implementation for the sample `getEmployeeSalary` interface, including the `processHeaders()` method that can handle incoming SOAP request headers of the form:

```
<SOAP-ENV:Header>
  <credentials>
    <username>scott</username>
    <password>tiger</password>
    <datasource>jdbc/OracleCoreDS</datasource>
  </credentials>
</SOAP-ENV:Header>
```

#### **Example 12-5 Employee Interface Extending HeaderCallback**

```
import oracle.j2ee.ws.HeaderCallback;
/**
 * Employee java class being exposed as Web Services
 * This service also extends HeaderCallback so as to
 * access Headers.
 */
public interface Employee
  extends HeaderCallback
{
  // Get the salary for a given Employee
  int getEmployeeSalary(String ename);
}
```

#### **Example 12-6 Including A HeaderCallback processHeaders() Implementation**

```
public void processHeaders(Header header)
  throws java.io.IOException,
         oracle.xml.parser.v2.XSLEException
{
  // Get all the Elements
  Vector entries = header.getHeaderEntries();
  Element e = (Element) entries.firstElement();
  System.out.println("Element received is: ");
  ((XMLElement)e).print(System.out);

  // Get independent nodes and retrieve node values.
  Node userNode;
  userNode = ((XMLNode)e).selectSingleNode("username");
  userName = ((XMLElement)userNode).getText();

  Node passwordNode;
  passwordNode = ((XMLNode)e).selectSingleNode("password");
```

```
password = ((XMLElement)passwordNode).getText();

Node dsNode;
dsNode = ((XMLNode)e).selectSingleNode("datasource");
datasourceName = ((XMLElement)dsNode).getText();

System.out.println("User name is: " + userName);
System.out.println("Password is: " + password);
System.out.println("Datasource is: " + datasourceName);
}
```

## Limitations for SOAP Header Support

The following list contains limitations related to SOAP header support:

1. Oracle Application Server Web Services does not provide support for processing or translating header information that is specified in a WSDL definition.
2. Oracle Application Server Web Services does not provide validation, XML or otherwise, for SOAP request header information provided in the `org.apache.soap.Header` object. The user is responsible for populating this object with well-formed XML.
3. Oracle Application Server Web Services does not provide support for SOAP response headers.
4. When proxies are generated for JMS Document Style Web Services, the SOAP request header `_setSOAPRequestHeaders()` and `_getSOAPRequestHeaders()` methods are not supplied. Using JMS Web Services there are no server-side facilities for processing SOAP request headers.
5. When proxies are generated for Stored Procedure Web Services, the SOAP request header `_setSOAPRequestHeaders()` and `_getSOAPRequestHeaders()` methods are not supplied. Using Stored Procedure Web Services there are no server-side facilities for processing SOAP request headers.

---

---

## Using Oracle Application Server SOAP

This appendix covers the following topics:

- [Understanding Oracle Application Server SOAP](#)
- [Apache SOAP Documentation](#)
- [Configuring the SOAP Request Handler Servlet](#)
- [Using OracleAS SOAP Management Utilities and Scripts](#)
- [Deploying OracleAS SOAP Services](#)
- [Using OracleAS SOAP Handlers](#)
- [Using OracleAS SOAP Audit Logging](#)
- [Using OracleAS SOAP Pluggable Configuration Managers](#)
- [Working With OracleAS SOAP Transport Security](#)
- [Using OracleAS SOAP Sample Services](#)
- [Using the OracleAS SOAP EJB Provider](#)
- [Using PL/SQL Stored Procedures With the SP Provider](#)
- [SOAP Troubleshooting and Limitations](#)
- [OracleAS SOAP Differences From Apache SOAP](#)
- [Apache Software License, Version 1.1](#)

## Understanding Oracle Application Server SOAP

In addition to the Oracle Application Server Web Services previously described in this chapter, that use a unique Servlet interface and J2EE deployment for Web Services, Oracle Application Server also provides Oracle Application Server SOAP (OracleAS SOAP) that is derived from Apache 2.3.1 SOAP and includes a number of enhancements.

The SOAP Message Processor, OracleAS SOAP, provides the following facilities:

- **SOAP Protocol Handling** - It provides an implementation of the interoperable SOAP specification. This includes support for Cookies and Sessions which is particularly useful to pass state information for stateful Web Services request/response.
- **Support for SOAP requests with Attachments (XML Payloads).**
- **Parsing** - OracleAS SOAP Processor integrates the Oracle XML Parser. For RPC-style requests, the OracleAS SOAP Processor can efficiently parse the incoming XML document, ensure the request is well-formed, and possibly validate the request. Similarly, it can also encode/serialize a Java response into a SOAP message.
- **Invoking Web Service Using Customized Web Services Servlet** - The SOAP Processor un-marshals the message contents and depending on the Servlet, calls the Web Services implementation. Web Services can be implemented as Java Classes, EJBs, or PL/SQL Stored Procedures.
- **Engaging a security manager to possibly authenticate the sender** - Before invoking the Web Services implementation, the OracleAS SOAP Processor (Servlet) authenticates the user using a standard JAAS-based User Manager plug-in. OracleAS SOAP Processor also supports Oracle's Single Sign-On Server and third-party authentication services to provide single-sign on for Web Services.
- **Exception Handling** - When exceptions occur during processing, the Java Exception is transformed to a SOAP fault and delivered to the service client.

## Apache SOAP Documentation

OracleAS SOAP is a modified version of Apache SOAP 2.3.1. Most of the documentation that applies to Apache SOAP 2.3.1 also applies to OracleAS SOAP. The Apache SOAP 2.3.1 documentation can be found at the following site:

<http://xml.apache.org/soap/docs/index.html>

## Configuring the SOAP Request Handler Servlet

The OracleAS SOAP Request Handler uses an XML configuration file to set required servlet parameters. By default, this file is named `soap.xml` and is placed in the `soap.ear` file in the directory `$SOAP_HOME/lib` on UNIX or `%SOAP_HOME%\lib` on Windows.

The XML namespace for this file is:

<http://xmlns.oracle.com/soap/2001/04/config>

To use a different configuration file for SOAP installation, expand the `soap.ear` file. In the directory `webapps/soap/WEB-INF` on UNIX or `webapps\soap\WEB-INF` on Windows, modify the path name specified for the `SoapConfig` parameter in the `soap.properties` file. Then, redeploy the updated `soap.ear` file.

For example, to change the configuration file from the default, `soap.xml`, to `newConfig.xml`, modify the value set for `soapConfig` in `soap.properties`.

```
servlet.soaprouter.initArgs=soapConfig=soap_home/soap/webapps/soap/WEB-INF/newConfig.xml
```

Where `soap_home` is the full path to the SOAP installation on your system.

The `pathAuth` boolean attribute, if set to `true`, enforces that clients must specify the unique service URL in order to post a message to the deployed service. The service URL is the SOAP servlet URL with the service URI appended on at the end. The default value of this attribute (if unspecified) is `false`.

[Table A-1](#) lists the SOAP Request Handler Servlet XML configuration file elements.

**Table A-1 SOAP Request Handler Servlet Configuration File Parameters**

Parameter	Description
<code>errorHandlers</code>	Specifies a list of handlers for the error handler chain.
<code>faultListeners</code>	This is an optional element that defines a list of <code>faultListener</code> elements. The <code>faultListener</code> element specifies a class that is invoked when a fault occurs. To cause a stack trace to be added to the SOAP fault that is returned to the user, specify a <code>faultListener</code> of <code>org.apache.soap.server.DOMFaultListener</code> .
<code>handler</code>	<p>The <code>handlers</code> element is an optional element that defines a list of handler elements. The handler element defines a global handler that can be configured to be invoked on every SOAP request in one of three contexts: request, response, error. You can define any number of handlers. The handler's <code>name</code> attribute specifies the name of the handler; each handler must have a unique name. The handler's <code>class</code> attribute specifies the Java class that implements the handler, and this class must implement the interface <code>oracle.soap.server.Handler</code>. Each handler may have any number of options, which are name-value pairs. The contexts are configured in the elements: <code>requestHandlers</code>, <code>responseHandlers</code>, and <code>errorHandlers</code>. Each of these elements defines an ordered list of handler names, or a chain of handlers.</p> <p>Note that SOAP creates one instance of each uniquely identified handler. Every appearance of a specific handler name in any chain refers to the same instance of the handler. Handlers are destroyed when the SOAP servlet is destroyed.</p>
<code>logger</code>	<p>Error and informational messages are logged using the class defined in the <code>logger</code> element. The logger class must extend <code>oracle.soap.server.Logger</code>.</p> <p>OracleAS SOAP includes the class <code>oracle.soap.server.impl.ServletLogger</code> that collects the servlet log methods so that SOAP messages are logged to the servlet log file. <code>ServletLogger</code> is the default logger. For the default logger, the severity option can be to any of the following values: <code>status</code>, <code>error</code>, <code>debug</code>.</p> <p>If you specify <code>error</code>, you will get both <code>status</code> and <code>error</code> messages. Similarly, if you specify <code>debug</code>, you will get all three types of messages.</p> <p>OracleAS SOAP includes two logger implementations. To log to the servlet log, use <code>oracle.soap.server.impl.ServletLogger</code>. To log to stdout, use <code>oracle.soap.server.impl.StdoutLogger</code>.</p> <p>You may implement your own logger by implementing the <code>oracle.soap.server.Logger</code> interface.</p>

**Table A-1 (Cont.) SOAP Request Handler Servlet Configuration File Parameters**

Parameter	Description
<code>providerManager</code>	<p>The <code>providerManager</code> is an optional element that allows a configuration manager to be defined. This defines how the server accesses provider deployment information.</p> <p>The <code>providerManager</code> class attribute specifies a Java class that implements <code>oracle.soap.server.ProviderManager</code>. The default configuration manager, <code>oracle.soap.server.impl.XMLProviderConfigManager</code>, persists the deployed providers to a file in XML format. It accepts a <code>filename</code> option. The filename is the path to the registry filename which may be a simple file name, relative path or an absolute path. If it is not an absolute path, then the path is determined from the filename and the servlet context. The default filename is <code>WEB-INF/providers.xml</code>.</p> <p>An alternative provider configuration manager, <code>oracle.soap.server.impl.BinaryProviderConfigManager</code>, persists the deployed providers in a file as a serialized object. The default file is <code>WEB-INF/providers.dd</code>.</p> <p>To specify a different configuration manager add a class attribute to the <code>configManager</code> element. For example:</p> <pre>&lt;osc:configManager class="fully.qualified.classname"&gt;</pre>
<code>requestHandlers</code>	Specifies a list of handlers for the request handler chain
<code>responseHandlers</code>	Specifies a list of handlers for the response handler chain
<code>serviceManager</code>	<p>The <code>serviceManager</code> is an optional element that allows a configuration manager to be defined and <code>ServiceManager</code> options to be set. This defines how the server accesses service deployment information. The <code>serviceManager</code> class attribute specifies a Java class that implements <code>oracle.soap.server.ServiceManager</code>.</p> <p>The default OracleAS SOAP configuration manager class is <code>oracle.soap.server.impl.XMLServiceConfigManager</code> which stores the service deployment information in an XML file. Using <code>XMLServiceConfigManager</code>, the file name is specified with the <code>filename</code> option. The filename is the path to the registry filename which may be a simple file name, relative path or an absolute path. If it is not an absolute path, then the path is determined from the filename and the servlet context. The default filename is <code>WEB-INF/services.xml</code>.</p> <p>To specify a different configuration manager add a class attribute to the <code>configManager</code> element.</p> <p>For example:</p> <pre>&lt;osc:configManager class="fully.qualified.classname"&gt;</pre> <p>An alternative service configuration manager, <code>oracle.soap.server.impl.BinaryServiceConfigManager</code>, persists the deployed services in a file as a serialized object. The default file is <code>WEB-INF/services.dd</code>.</p> <p>The service manager can automatically deploy the provider manager and the service manager as SOAP services. To allow these managers to be exposed as services, set the <code>autoDeploy</code> option to <code>true</code>. By default <code>autoDeploy</code> value is <code>false</code>.</p>

## Using OracleAS SOAP Management Utilities and Scripts

To use the OracleAS SOAP management utilities, you need to set up the execution environment for executing SOAP management utilities using one of the supplied client side scripts. The `clientenv` scripts set the `CLASSPATH` and add the `$SOAP_HOME/bin` directory to the path.

To set the client environment, on UNIX, use the following commands:

```
cd $SOAP_HOME/bin
source clientenv.csh
```

On Windows, use the following commands:

```
cd %SOAP_HOME%\bin
```

```
clientenv.bat
```

The `clientenv` scripts sets environment variables that are used by the other scripts and the samples. You can override these by setting the environment variables yourself. The variable `SOAP_URL` is the URL of the SOAP server and `JAXP` is set to use the `DocumentBuilderFactory` for the Oracle XML parser.

## Managing Providers

The `providerMgr` script runs the SOAP client that manages providers. Run the script without any parameters for usage information.

On UNIX, use the following command:

```
providerrMgr.sh options
```

On Windows, use the following command:

```
providerMgr.bat options
```

Where the *options* for `providerMgr` are:

```
deploy ProviderDescriptorFile
```

This deploys the provider described in the *ProviderDescriptorFile* and makes the provider available.

```
undeploy ProviderID
```

This removes the provider with the supplied *ProviderID*. The *ProviderID* is the `id` attribute specified in the provider descriptor file.

The Java provider is deployed once at installation time with `id=java-provider`, but any provider you create must be explicitly deployed. For example, on UNIX, to deploy a provider using the provider deployment descriptor `provider.xml`, use the following command:

```
providerMgr.sh deploy provider.xml
```

## Using the Service Manager to Deploy and Undeploy Java Services

The `ServiceMgr` is an administrative utility that deploys and undeploys SOAP services. To deploy a service, first set the SOAP environment, then use the `deploy` command. On UNIX, the command is:

```
source clientenv.csh
ServiceMgr.sh deploy ServiceDescriptorFile
```

For Windows, the command is:

```
clientenv.bat
ServiceManager.bat deploy ServiceDescriptorFile
```

The `deploy` option makes the service specified in `ServiceDescriptorFile` available.

When you are ready to undeploy a service, use the `undeploy` command with the registered service name as an argument. On UNIX, the command is:

```
ServiceManager.sh undeploy ServiceID
```

For Windows, the command is:

```
ServiceManager.bat undeploy ServiceID
```

This makes the service with the given id unavailable. The *ServiceID* is the service id attribute specified in the service descriptor file.

The *ServiceMgr* supports listing and querying SOAP services. To list the available services, first set the SOAP environment, then use the `list` command. On UNIX, the command is:

```
source clientenv.csh
ServiceMgr.sh list
```

On Windows, the command is:

```
clientenv.bat
ServiceMgr.bat list
```

To query a service and obtain the descriptor parameters set in the service deployment descriptor file, use the `query` command. On UNIX, the command is:

```
ServiceMgr.sh query ServiceID
```

On Windows, the command is:

```
ServiceMgr.bat query ServiceID
```

Where *ServiceID* is the service id attribute set in the service descriptor file.

## Generating Client Proxies from WSDL Documents

The `wsd12java` script takes as input a WSDL document and returns a Java class which can be used to call the service. The Java class contains methods with the same names as those described in the WSDL document. The generated code make calls to the Apache client side libraries.

On UNIX, use the following command:

```
wsd12java.sh options
```

On Windows, use the following command:

```
wsd12java.bat options
```

Where the *options* for `wsd12java` are:

```
wsd12java . sh WsdldocumentURL OutputDir [-k PackageName] [-s ServiceName] [-p PortName]
```

Where:

*WsdldocumentURL* is the URL of the WSDL document.

*OutputDir* is the output directory for generated proxy Java code.

*-k PackageName* is the package name for generated proxy Java code.

*-s ServiceName* is the service name for which proxy will be generated.



-p *PortName* the port name of the service. The proxy is generated for the specified port of the service.

The output directory structure is:

*output root dir/ service name/ port name/ package name/ java proxy source code*

By default, the *PackageName* will be the same as the WSDL service name.

If neither of -s and -p options is specified, proxies for all ports of all services are generated. Without -p option specified, proxies for all ports of the specified service are generated.

## Generating WSDL Documents from Java Service Implementations

The `java2wsdl` script takes as input a Java class and creates as output a WSDL document describing the class as an RPC service. When the Java class is used as a Web Service, the associated WSDL document can be transmitted to developers who might wish to call the service.

On UNIX, use the following command:

```
java2wsdl.sh options
```

On Windows, use the following command:

```
java2wsdl.bat options
```

Where the *options* for `wsdl2java` are:

```
java2wsdl.sh ClassName OutputFile SoapURL ClassURL1 ClassURL2 ...
```

Where:

*ClassName* is the fully qualified path name of a Java `.class` file that is to be a Web Service.

*OutputFile* is the output WSDL document name.

*SoapURL* is the SOAP endpoint.

*ClassURL* list serves as a class path for searching referenced classes

## Deploying OracleAS SOAP Services

This section covers the following topics related to deploying and undeploying OracleAS SOAP Services:

- [Creating Deployment Descriptors](#)
- [Installing a SOAP Web Service in OC4J](#)
- [Disabling an Installed SOAP Web Service](#)
- [Installing a SOAP Web Service in an OC4J Cluster](#)

## Creating Deployment Descriptors

Deployment descriptors include service deployment descriptors and provider deployment descriptors. A provider deployment descriptor file is an XML file that describes, to the SOAP servlet, the configuration information for a provider. A service deployment descriptor file is an XML file that describes, to the SOAP servlet, the configuration information for a service.

Services written in Java only require a service descriptor. All Java service descriptors may point to the same Java provider descriptor supplied with the OracleAS SOAP installation.

Each service written as a PL/SQL stored procedure requires one service descriptor and one provider descriptor for each database user. The advantage of this is that when a password or user is changed, only one descriptor needs to be updated, not every service descriptor.

See the Stored Procedure section for more information.

Services written as an EJB require one service descriptor and one provider descriptor for each EJB container user.

See the EJB section of this document for more information.

---

---

**Note:** For developers who wish to write their own providers, the Apache style provider interface and descriptors are also supported. Apache descriptors contain both service and provider properties in a single file, so common provider information must be duplicated for every service.

---

---

A service deployment descriptor file defines the following information:

- The service ID
- The service provider type (for example, Java)
- The available methods

The best way to write a descriptor is to start with a copy of an existing descriptor from one of the sample directories.

[Example A-1](#) shows the Java `SimpleClock` service descriptor file `SimpleClockDescriptor.xml`. This descriptor file is included in the `samples/simpleclock` directory. The service descriptor file must conform to the service descriptor schema (the schema, `service.xsd`, is located in the directory `$SOAP_HOME/schemas` on UNIX or in `%SOAP_HOME%\schemas` on Windows).

The service descriptor file identifies methods associated with the service in the `isd:provider` element that uses the `methods` attribute. The `isd:java class` element identifies the Java class that implements the SOAP service, and provides an indication of whether the class is static.

### **Example A-1 Java Service Descriptor File for Sample Simple Clock Service**

```
<isd:service xmlns:isd="http://xmlns.oracle.com/soap/2001/04/deploy/service"
            id="urn:jurassic-clock"
            type="rpc" >
  <isd:provider
    id="java-provider"
```

```

        methods="getDate"
        scope="Application" >
        <isd:java class="samples.simpleclock.SimpleClockService" />
    </isd:provider>
    <!-- includes stack trace in fault -->
    <isd:faultListener class="org.apache.soap.server.DOMFaultListener" />
</isd:service>

```

---

**Note:** The service descriptor file does not define the method signature for service methods. SOAP uses reflection to determine method signatures.

---

## Installing a SOAP Web Service in OC4J

Install an OracleAS SOAP Web Service in Oracle Application Server Containers for J2EE (OC4J) by performing the following steps:

1. Create service and provider deployment descriptors.
2. Expand the `soap.ear` file found in `$SOAP_HOME/lib` on UNIX or `%SOAP_HOME\lib` on Windows.
3. Copy Java classes and Jars implementing the service to the correct locations in the expanded `soap.ear` directories.

Copy Java `.class` files to `WEB-INF/classes`.

Copy Java `.jar` files to `WEB-INF/libs`.

4. Redeploy the updated `soap.ear` file.
5. Deploy the provider descriptor by executing the command:

```
providerMgr.sh deploy FileName
```

where *FileName* is the name of the provider descriptor xml file.

6. Deploy the service by executing the command:

```
serviceMgr.sh deploy FileName
```

Where *FileName* is the name of the service descriptor xml file.

## Disabling an Installed SOAP Web Service

To disable an installed service, run the command:

```
serviceMgr.sh undeploy ServiceID
```

where *ServiceID* is the id attribute of the service element in the service descriptor.

## Installing a SOAP Web Service in an OC4J Cluster

It is necessary to install an OracleAS SOAP service on every machine in a cluster. If the service is not installed on all machines in a cluster, the cluster dispatcher might dispatch a service request to a machine that does not have the service, resulting in an error on the service invocation.

## Using OracleAS SOAP Handlers

A handler is a class that implements the `oracle.soap.server.Handler` interface. A handler can be configured as part of a chain in one of three contexts: request, response, or error. Note that handlers in a chain are invoked in the order they are specified in the configuration file.

### Request Handlers

Handlers in the request chain are invoked on every request that arrives, immediately after the SOAP Request Handler Servlet reads the SOAP Envelope. If any handler in the request chain throws an exception, the processing of the chain is immediately terminated and the service is not invoked.

The error chain is invoked if any exception occurs during request chain invocation.

### Response Handlers

Handlers in the response chain are invoked on every request immediately after the service completes. If any handler in the response chain throws an exception, processing of the chain is immediately terminated. The error chain is invoked if any exception occurs during response chain invocation.

### Error Handlers

When an exception occurs during either request-chain invocation, service invocation, or response-chain invocation, the SOAP Request Handler Servlet invokes the handlers in the error chain. In contrast to the request and response chains, an exception from an error handler is logged and processing of the error chain continues. All handlers in the error chain are invoked, regardless of whether one of the error handlers throws an exception.

## Configuring Handlers

Configure handlers and handler chains in the SOAP configuration file. Handlers can be invoked for each service request or response, or when an error occurs. Handlers are global in the sense that they apply to every SOAP request and cannot be configured on a subset of requests, such as all requests for a particular service.

Configure a handler by setting parameters in the SOAP configuration file, `soap.xml`. [Example A-2](#) shows a sample segment from a SOAP configuration file showing the configuration for a handler.

#### **Example A-2 Handler Configuration**

```
<osc:handlers>
  <osc:handler name="auditor"
    class="oracle.soap.handlers.audit.AuditLogger">
    <osc:option name="auditLogDirectory"
      value="/private1/oracle/app/product/tv02/soap/webapps/soap/WEB-INF"/>
    <osc:option name="filter" value="(!(host=localhost))"/>
  </osc:handler>
</osc:handlers>

<osc:requestHandlers names="auditor"/>
<osc:responseHandlers names="auditor"/>
<osc:errorHandlers names="auditor"/>
```

## Using OracleAS SOAP Audit Logging

The OracleAS SOAP audit logging feature monitors and records SOAP usage. Audit logging maintains records for postmortem analysis and accountability. The SOAP audit logging feature complements the audit logging capabilities available with the OC4J server which hosts the SOAP Request Handler Servlet (SOAP server).

OracleAS SOAP stores audit trails as XML documents. Using XML documents, OracleAS SOAP creates portable audit trails and enables the transformation of complete audit trails or individual audit records to different formats.

By default, OracleAS SOAP audit logging uses an audit logger class that implements the `Handler` interface (part of the `oracle.soap.server` package). The audit logger class is invoked conditionally to monitor events including service requests, service responses, and errors.

This section covers the following topics:

- [Audit Logging Information](#)
- [Auditable Events](#)
- [Configuring the Audit Logger](#)

### Audit Logging Information

[Table A-2](#) lists the audit logging elements available for each audit log record. Individual audit log records may not contain all these elements. In the log file, each audit log record is stored as a `SoapAuditRecord` element.

**Table A-2 Auditable Audit Record Elements**

Audit Record Element	Description
HostName	Specifies the hostname of the client that sent the request.
IpAddress	Specifies the IP address of the client that sent the request.
Method	Specifies the method name for the SOAP request.
Request Envelope	Provides the complete SOAP request message.
Request Envelope Method	Name of the Method in the SOAP request envelope
Request Envelope URI	Specifies the URI of the service in the SOAP request envelope.
Response Envelope	Provides the complete SOAP response message.
ServiceURI	Specifies the service URI for the SOAP request.
SoapAuditRecord	Contains an individual record. The <code>chainType</code> attribute indicates if the record is generated as part of a request or a response.
TimeStamp	Specifies the system time when the SOAP audit record was generated.
User	Specifies the username associated with the request. Note, this element is only provided when a user context is associated with the service request or service response.

### Audit Logging Output

The XML schema for the generated audit log is provided in the file `SoapAuditTrail.xsd` in the directory `$SOAP_HOME/schema` on UNIX or `%SOAP_HOME%\schema` on Windows. Refer to the schema file for complete details on the format of a generated audit log record.

## Auditable Events

The audit logger class is invoked when an auditable event occurs and the SOAP Request Handler Servlet is configured to enable auditing for the event. Auditable events include a service request or a service response.

### Audit Logging Filters

An audit logging filter can be specified in the SOAP configuration file to limit the set of auditable events that are recorded to the audit log. The SOAP server applies event filters to request and response events. [Table A-4](#) shows the filter attributes available to select with an event filter specification. When applied, filters limit the number of records generated in the audit log. For example, when a filter is specified for a particular host, only the auditable events generated for the specified host are saved to the audit log.

The syntax for defining auditable events with a filter is derived from RFC 2254. [Table A-3](#) shows the filter syntax, and [Example A-3](#) provides several examples.

#### See Also:

- ["Configuring the Audit Logger"](#) on page A-13
- <ftp://ftp.isi.edu/in-notes/rfc2254.txt> on RFC 2254

**Table A-3** Audit Trail Events Filter Attributes

Audit Event Filter Attributes	Description
Host	<p>Specifies the hostname of the host for the service request or response. If this attribute is not specified in a filter, the hostname of the client is not used in filtering audit log records.</p> <p>Fully specify the hostname of the client or use wildcards ("*"). Wildcards embedded within the specified hostname are not supported the examples show valid and invalid uses of wildcards. If a wildcard is used then the wildcard must be the first character in the filter. Case is ignored for hostnames. Care should be used in setting this attribute. Depending on the DNS setup, the hostname returned could be fully qualified or nonqualified; for example, <code>explosives.acme.com</code> or <code>explosives</code>. For some IP addresses, the DNS may not be able to resolve the hostname.</p> <p>Legal values for a <code>Host</code> filter attribute include the following examples:  <code>explosives.acme.com</code>, <code>*.acme.com</code>, <code>*.com</code></p> <p>Illegal values for a <code>Host</code> filter attribute include the following examples:  <code>*</code>, <code>explosives.acme.*</code>, <code>explosives.*</code>, <code>ex*s.acme.com</code>, <code>*ives.acme.com</code></p>
ip	<p>Specifies the IP address of the client for the service request or response.</p> <p>The IP address of the client has to be either fully specified, using all four bytes, in the dot separate decimal form, or specified using wildcards ("*"). Embedded wildcards are not supported. If a wildcard is used then the wildcard must be the last character in the filter.</p> <p>If this attribute is not used in a filter then the IP address of the client is not used in filtering.</p> <p>Legal values for an <code>ip</code> filter attribute include the following examples:  <code>138.2.142.154</code>, <code>138.2.142.*</code>, <code>138.2.*</code>, <code>138.*</code></p> <p>Illegal values for an <code>ip</code> filter attribute include the following examples:  <code>*</code>, <code>138.2.*.154</code>, <code>*.2</code>, <code>138.*.152</code>, <code>138.2.142</code>, <code>138.2</code>, <code>138</code></p>
urn	Specifies the service URN. Wildcards are not supported for this attribute.
username	<p>Specifies the transport level username associated with the client.</p> <p>Wildcards are not supported in a username filter attribute.</p>

**Table A-4 Audit Log Filter Syntax**

Filter Value	Description
attr	1*(any US-ASCII char except "*", "(", ")", "&", " ", "!", "*", "=")
equal	"="
filter	"( <i>filtercomp</i> )" Whitespaces between "( <i>filtercomp</i> and ")" are not allowed.
filtercomp	<i>and</i>   <i>or</i>   <i>not</i>   <i>item</i> <i>and</i> = "&" <i>filterlist</i> <i>or</i> = " " <i>filterlist</i> <i>not</i> = "!" <i>filter</i>
filterlist	2*2 <i>filter</i>
filtertype	<i>equal</i>
item	<i>attr filtertype value</i> Whitespaces between <i>attr</i> , <i>filtertype</i> and <i>value</i> are not allowed.
value	1*(any octet except ASCII representation of ") - 0x29). The character "*" has a special meaning. The "*" character is referred to as a wildcard and matches anything.

**Example A-3 Sample Audit Log Filters**

```
(ip=138.2.142.154)
(! (host=localhost))
(! (host=*.acme.com))
(&(host=*.acme.com)(username=daffy))
(&(ip=138.2.142.*) (| (urn=urn:www-oracle-com:AddressBook)(username=daffy)))
```

## Configuring the Audit Logger

Configure the default SOAP audit logger supplied with Oracle Application Server by setting parameters in the SOAP configuration file, `soap.xml`. To enable the default audit logger and turn on audit logging, do the following in the configuration file.

- Define the name and options for the audit log handler. The default SOAP audit logger is defined in the class `oracle.soap.handlers.audit.AuditLogger`. The default audit logger supports several options that you specify in the configuration file. [Table A-5](#) shows the available audit logger options.
- Add the name for the audit logger handler to the `requestHandler`, `responseHandler`, or `errorHandler` chain (or to all of the handler chains).

[Example A-4](#) shows a sample segment from a SOAP configuration file including the audit logging configuration options. [Example A-4](#) shows configuration options set to use all options. However, this configuration would produce an extremely large audit log, and is not recommended.

---

**Note:** When you audit errors using the audit logger, depending on when the error occurs in the request-chain or the response-chain, it is possible that the request or response message may not be included in the audit log record, even with `includeRequest` or `includeResponse` enabled.

---

#### Example A-4 Audit Logging Configuration

```
<osc:handlers>
  <osc:handler name="auditor"
    class="oracle.soap.handlers.audit.AuditLogger">
    <osc:option name="auditLogDirectory"
      value="/private1/oracle/app/product/tv02/soap/webapps/soap/WEB-INF"/>
    <osc:option name="filter" value="(!(host=localhost))"/>
    <osc:option name="includeRequest" value="true"/>
    <osc:option name="includeResponse" value="true"/>
  </osc:handler>
</osc:handlers>
<osc:requestHandlers names="auditor"/>
<osc:responseHandlers names="auditor"/>
<osc:errorHandlers names="auditor"/>
```

**Table A-5 Audit Logger Configuration Options**

Option	Description
<code>auditLogDirectory</code>	Specifies the directory where the audit log file is saved. The <code>auditLogDirectory</code> option is required. The name of the generated audit log file is <code>OracleSoapAuditLog.timestamp</code> , where <code>timestamp</code> is the date and time the file is first generated.  <b>Valid values:</b> any string that is a valid directory
<code>filter</code>	Specifies the audit event filter. This option is optional. If a <code>filter</code> is not specified SOAP server logs every event.  <b>Valid values:</b> any valid filter.
<code>includeRequest</code>	Specifies that the audit record include the request message for the event that generated the audit log record.  <b>Valid values:</b> <code>true</code> , <code>false</code> Any value other than <code>true</code> or <code>false</code> is treated as an error. <b>Default Value:</b> <code>false</code>
<code>includeResponse</code>	Specifies that the audit record include the response message for the event that generated the audit log record.  <b>Valid values:</b> <code>true</code> , <code>false</code> Any value other than <code>true</code> or <code>false</code> is treated as an error. <b>Default Value:</b> <code>false</code>

**See Also:** ["Using OracleAS SOAP Handlers"](#) on page A-10



## Using OracleAS SOAP Pluggable Configuration Managers

OracleAS SOAP supports pluggable configuration managers similar to those supported in Apache SOAP 2.3.1. Since OracleAS SOAP supports provider deployment descriptors separate from service deployment descriptors, the interface details using OracleAS SOAP are slightly different from Apache SOAP 2.3.1. In OracleAS SOAP, configuration managers are configured separately for the provider manager and the service manager. All configuration managers must implement the `oracle.soap.server.ConfigManager` interface.

To simplify development, when you write a configuration manager implementation, you may use the abstract class that is provided with OracleAS SOAP (`oracle.soap.server.impl.BaseConfigManager`). This abstract class provides a standard implementation for most of the `ConfigManager` interface with two abstract methods that read and write the persistent store.

[Example A-5](#) shows a sample implementation of a provider configuration manager.

### **Example A-5 Sample Provider Configuration Manager Implementation.**

```
public class MyProviderConfigManager extends BaseConfigManager
{
    public void setOptions(Properties options)
        throws SOAPException
    {
        // handle implementation specific options
    }

    public void readRegistry()
        throws SOAPException
    {
        // read the deployed providers from persistent store
    }

    public void writeRegistry()
        throws SOAPException
    {
        // write the deployed providers to persistent store
    }
}
```

The `setOptions` method is passed the options specified in any `<option>` elements specified in the `<configManager>` element. Synchronization of reading/writing the registry is the responsibility of the specific configuration manager implementation.

## Working With OracleAS SOAP Transport Security

Oracle Application Server uses the security capabilities of the underlying transport that sends SOAP messages. Oracle Application Server supports the HTTP and HTTPS protocols for sending SOAP messages. HTTP and HTTPS support the following security features:

- HTTP proxies
- HTTP authentication (basic RFC 2617)
- Proxy authentication (basic RFC 2617)

OracleAS SOAP Client transport uses the modified, to support Oracle Wallet Manager, `HTTPClient` package. OracleAS SOAP transport defines several properties to support

these features. [Table A-6](#) lists the client-side security properties that Oracle Application Server supports.

In an OracleAS SOAP Client application, you can set the security properties shown in [Table A-6](#) as system properties by using the `-D` flag at the Java command line. You can also set security properties in the Java program by adding these properties to the system properties (use `System.setProperty()` to add properties).

[Example A-6](#) shows how Oracle Application Server supports overriding the values specified for system properties using Oracle Application Server transport specific APIs. The `setProperty()` method in the class `OracleSOAPHTTPConnection` contains set properties specifically for the HTTP connection (this class is in the package `oracle.soap.transport.http`).

**Example A-6 Setting Security Properties for OracleSOAPHTTPConnection**

```
org.apache.soap.rpc.Call call = new org.apache.soap.rpc.Call();
oracle.soap.transport.http.OracleSOAPHTTPConnection conn =
(oracle.soap.transport.http.OracleSOAPHTTPConnection) call.getSOAPTransport();
java.util.Properties prop = new java.util.Properties();
// Use client code to set name-value pairs of properties in prop
.
.
.
conn.setProperties(prop);
```

---



---

**Note:** The property `java.protocol.handler.pkgs` must be set as a system property.

---



---

**Table A-6 SOAP HTTP Transport Security Properties**

Property	Description
<code>http.authRealm</code>	Specifies the realm for which the HTTP authentication username/password is specified. This property is mandatory when using basic authentication.
<code>http.authType</code>	Specifies the HTTP authentication type. The case of the value specified is ignored. Valid values: <code>basic</code> , <code>digest</code> The value <code>basic</code> specifies HTTP basic authentication. Specifying any value other than <code>basic</code> or <code>digest</code> is the same as not setting the property.
<code>http.password</code>	Specifies the HTTP authentication password.
<code>http.proxyAuthRealm</code>	Specifies the realm for which the proxy authentication username/password is specified.
<code>http.proxyAuthType</code>	Specifies the proxy authentication type. The case of the value specified is ignored. Valid values: <code>basic</code> , <code>digest</code> Specifying any value other than <code>basic</code> or <code>digest</code> is the same as not setting the property.
<code>http.proxyHost</code>	Specifies the hostname or IP address of the proxy host.
<code>http.proxyPassword</code>	Specifies the HTTP proxy authentication password.

**Table A-6 (Cont.) SOAP HTTP Transport Security Properties**

Property	Description
<code>http.proxyPort</code>	<p>Specifies the proxy port. The specified value must be an integer. This property is only used when <code>http.proxyHost</code> is defined; otherwise this value is ignored.</p> <p>Default value: 80</p>
<code>http.proxyUsername</code>	Specifies the HTTP proxy authentication username.
<code>http.username</code>	Specifies the HTTP authentication username.
<code>java.protocol.handler.pkgs</code>	<p>Specifies a list of package prefixes for <code>java.net.URLStreamHandlerFactory</code>. The prefixes should be separated by " " vertical bar characters.</p> <p>This value should contain: <code>HTTPClient</code></p> <p>This value is required by the Java protocol handler framework; it is not defined by Oracle Application Server. This property must be set when using HTTPS. If this property is not set using HTTPS, a <code>java.net.MalformedURLException</code> is thrown.</p> <p><b>Note:</b> This property must be set as a system property.</p> <p>For example, set this property as shown in either of the following:</p> <ul style="list-style-type: none"> <li>■ <code>java.protocol.handler.pkgs=HTTPClient</code></li> <li>■ <code>java.protocol.handler.pkgs=sun.net.www.protocol   HTTPClient</code></li> </ul>
<code>oracle.soap.transport.1022ContentType</code>	<p>Specifies the value for the Content-Type HTTP header in Oracle9iAS, and in Oracle Application Server 10g. The value for this property supports Oracle SOAP servers running either Oracle 9iAS Release 1.0.2.2 or Release 9.0.x or 10g (9.0.4). This property provides interoperability between Oracle9iAS Release 9.0.x Oracle SOAP clients or Oracle Application Server 10g (9.0.4) and older server versions (as distributed with Oracle9iAS Release 1.0.2.2).</p> <p>Valid values: <code>true</code>, <code>false</code> (case is ignored)</p> <p>Setting the value to <code>true</code> specifies to use the Oracle9 iAS Release 1.0.2.2 content-type HTTP header values when the SOAP message is sent. In this case, the value is set to: <code>content-type: text/xml</code></p> <p>Setting the value to <code>false</code> specifies to use the Oracle Application Server version 9.0.x content-type header value when the SOAP message is sent. In this case, the value is set to: <code>content-type: text/xml; charset=utf-8</code></p> <p>The value <code>false</code> is the default value.</p> <p><b>Note:</b> for SOAP messages with attachments, the content-type HTTP header is always set to the value: <code>multipart/related</code>.</p>
<code>oracle.soap.transport.allowUserInteraction</code>	<p>Specifies the allows user interaction parameter. The case of the value specified is ignored. When this property is set to <code>true</code> and either of the following are true, the user is prompted for a username and password:</p> <ol style="list-style-type: none"> <li>1. If any of properties <code>http.authType</code>, <code>http.username</code>, or <code>http.password</code> is not set, and a 401 HTTP status is returned by the HTTP server.</li> <li>2. If either of properties <code>http.proxyAuthType</code>, <code>http.proxyUsername</code>, or <code>http.proxyPassword</code> is not set and a 407 HTTP response is returned by the HTTP proxy.</li> </ol> <p>Valid values: <code>true</code>, <code>false</code></p> <p>Specifying any value other than <code>true</code> is considered as <code>false</code>.</p>

**Table A–6 (Cont.) SOAP HTTP Transport Security Properties**

Property	Description
oracle.ssl.ciphers	Specifies a list of: separated cipher suites that are enabled. Default value: The list of all cipher suites supported by Oracle SSL are supported.
oracle.wallet.location	Specifies the location of an exported Oracle wallet or exported trustpoints. Note: The value used is not a URL but a file location, for example: /etc/ORACLE/Wallets/system1/exported_wallet (on UNIX) d:\oracle\system1\exported_wallet (on Windows) This property must be set when HTTPS is used with SSL authentication, server or mutual, as the transport.
oracle.wallet.password	Specifies the password of an exported wallet. Setting this property is required when HTTPS is used with client, mutual authentication as the transport.

## Apache Listener and Servlet Engine Configuration for SSL

When using Apache listener and `mod_ssl` (or `mod_ossl`), the following directives must be set for the `soap servletlocation/directory`:

```
SSLOption +StdEnvVars +ExportCertData
```

This directive can be set conditionally, refer to `mod_ssl/mod_ossl` documentation for details. By default this directive is disabled for performance reasons. If this directive is not set then the servlet engine does not have a way to access the SSL related data (such as the cipher suite, client cert etc).

## Using JSSE with Oracle Application Server SOAP Client

This section describes how to use SSL with the OracleAS SOAP Client side when the Oracle security infrastructure is not available. Availability of Oracle security infrastructure means the availability of Oracle client side libraries (including `$ORACLE_HOME/lib/*`, `$ORACLE_HOME/jlib/javax-ssl-1_2.jar`, and `$ORACLE_HOME/jlib/jssl-1_2.jar`).

OracleAS SOAP uses the following class as the default transport class:

```
oracle.soap.transport.http.OracleSOAPHTTPConnection
```

This class uses a modified version of `HTTPClient` package. For information on `HTTPClient`, see the following site:

<http://www.innovation.ch/java/HTTPClient/>

This version of `HTTPClient` package is integrated with Oracle Java SSL and supports Oracle Wallet for HTTPS transport. If a SOAP client side does not have OracleAS SOAP Client side available, it is still possible to use HTTPS as a transport with OracleAS SOAP Client side libraries.

To do this, follow these steps:

1. Use the following transport class:

```
class org.apache.soap.transport.http.SOAPHTTPConnection
```

If using RPC then call the following method by passing an instance of `org.apache.soap.transport.http.SOAPHTTPConnection` as an argument:

```
method org.apache.soap.rpc.Call#setSOAPTransport
(org.apache.soap.transport.SOAPTransport)
```

For example:

```
org.apache.soap.rpc.Call myCallObj = new
org.apache.soap.rpc.Call();
myCallObj.setSOAPTransport(new
org.apache.soap.transport.http.SOAPHTTPConnection());
```

If using messaging, then call the following method by passing an instance of `org.apache.soap.transport.http.SOAPHTTPConnection` as an argument:

```
org.apache.soap.messaging.Message#setSOAPTransport
(org.apache.soap.transport.SOAPTransport)
```

For example:

```
org.apache.soap.messaging.Message myMsgObj = new
org.apache.soap.messaging.Message();
myMsgObj.setSOAPTransport(new
    org.apache.soap.transport.http.SOAPHTTPConnection());
```

2. Download Java Secure Socket Extension (JSSE) and configure JSSE according to the supplied instructions. JSSE is available at the following site:

<http://java.sun.com/products/jsse/>

- Make sure the files `jnet.jar`, `jcrt.jar` and `jsse.jar` are in the classpath or in the installed extensions directory (`$JRE_HOME/lib/ext`).
- Make sure that SunJSSE provider is correctly configured. This can be done either statically by editing the `$JRE_HOME/lib/security/java.security` file and adding the line:

```
security.provider.num=com.sun.net.ssl.internal.ssl.Provider
```

Where *num* is 1-based preference order or by dynamically by adding the provider at run time by adding the following line of code:

```
Security.addProvider(new com.sun.net.ssl.internal.ssl.Provider());
```

Dynamic addition of security providers requires that appropriate permissions are set.

- Make sure the system property `java.protocol.handler.pkgs` is set to `com.sun.net.ssl.internal.www.protocol`
- If using proxy server, make sure that the following system properties are set to the correct proxy hostname and proxy port, respectively:

```
https.proxyHost
https.proxyPort
```

- If using SSL with server side authentication and the default `TrustManager`, ensure that the certificate signer of the server is one of the following files:

```
$JRE_HOME/lib/security/jssecacerts
```

or if `jssecacerts` does not exist:

`$JRE_HOME/lib/security/cacerts`

- To override the KeyManager/TrustManager keystore default locations, use the system properties:

```
javax.net.ssl.keystore
javax.net.ssl.keyStoreType
javax.net.ssl.keyStorePassword
javax.net.ssl.trustStore
javax.net.ssl.trustStoreType
javax.net.ssl.trustStorePassword
```

Please consult JSSE documentation for details. If using a specific third party JSSE implementation, please consult the appropriate documentation.

**See Also:** HTTPClient information at the site:

<http://www.innovation.ch/java/HTTPClient/>

## Using OracleAS SOAP Sample Services

The section lists the samples included with OracleAS SOAP. The class files for all of the samples are in `$SOAP_HOME/lib/samples.jar` on UNIX or in `%SOAP_HOME%\lib\samples.jar` on Windows.

To run any sample, you need to ensure that `samples.jar` is available on your servlet's CLASSPATH. Please refer to the README included with each sample for more information.

### The Xmethods Sample

The clients in the xmethods sample represent the easiest way to get started with SOAP because they are clients that access existing services that are hosted on systems on the internet. Information on these services can be found at the site:

<http://www.xmethods.org>

This sample is in `$$SOAP_HOME/samples/xmethods`.

### The AddressBook Sample

This sample has a service implemented in Java and several clients. This sample illustrates literal XML encoding. See `$$SOAP_HOME/samples/addressbook` for the sample source code. This directory also contains a script that illustrates how to run the sample addressbook clients using HTTPS as transport.

### The StockQuote Sample

This sample has a service implemented in Java and one client. It is located in `$$SOAP_HOME/samples/stockquote`

### The Company Sample

This sample has a service that is comprised of PL/SQL stored procedures and several clients. It is located in `$$SOAP_HOME/samples/sp/company`. Check the README file in this directory for details on how to setup, compile, and test this sample service.

## The Provider Sample

This includes a template provider that can be used as a starting point for creating your own provider.

## The AddressBook2 Sample

This sample demonstrates use of the Addressbook service with session scope. It shows how to maintain the same HTTP session across SOAP Calls. It contains an example of a SOAP client proxy generated from a WSDL service description file. It is located in `$$SOAP_HOME/samples/addressbook2`

## The Messaging Sample

This sample is an example of a message-based SOAP service. It is located in `$$SOAP_HOME/samples/messaging`

## The Mime Sample

This sample does SOAP with attachments using both RPC and message based services. It is located in `$$SOAP_HOME/samples/mime`.

## Using the OracleAS SOAP EJB Provider

This section compares the OracleAS SOAP EJB providers with the Apache-SOAP 2.2 EJB providers.

### Stateless Session EJB Provider

In Apache SOAP, the Stateless EJB provider, on receiving the SOAP request, performs a JNDI lookup on the home interface of the EJB. The Stateless EJB provider then invokes a create on the EJB's Home Interface in order to get a reference to a stateless EJB. Then it uses this EJB reference to invoke the requested method.

OracleAS SOAP uses the same mechanism to support Stateless Session EJBs as Apache SOAP.

### Stateful Session EJB Provider in Apache SOAP

On receiving a first time SOAP request, the Apache SOAP Stateful Session EJB provider first locates the Home Interface through a JNDI lookup and using a subsequent create obtains an object reference to a Stateful Session EJB. The provider then invokes the requested method on the object reference.

In the next step the provider serializes the `EJBHandle` of the specified EJB reference and appends it to the targetURI with an "@" delimiter. The Stateful Session EJB provider then sends this modified target URI back to the requesting SOAP client. If the client wants to reuse the same EJB instance, it must retrieve this "modified" target URI for the service from the Response and set it in the next SOAP Call.

Upon receiving this request, the Stateful EJB provider extracts the stringified EJB reference and deserializes it into an `EJBHandle` from which it can obtain the EJB reference. It can then invoke the method on the specified EJB.

The drawback of the Apache SOAP implementation is that the client must be EJB aware and that it could not operate with other SOAP servers.

OracleAS SOAP offers an alternative solution for Stateful Session EJBs that allows for client interoperability.

## Stateful Session EJB Provider in OracleAS SOAP

The OracleAS SOAP Stateful Session EJB provider binds the EJB reference to the current session, if none is bound, otherwise, it merely retrieves the EJB reference from the session. In order for the client to access the same Stateful Session EJB, the client has to simply maintain its current session between successive calls.

If at any point in a session, the SOAP client invokes a create on the EJB's Home Interface, the provider binds the EJB reference from the create to the session, to be used for other call requests within the session.

## Entity EJB Provider in OracleAS SOAP

In order for a SOAP client to run a business method on an entity EJB, it first needs to either "create" a new EJB upon which to run the method or *find* an already existing EJB which suit some criteria. Access to an entity EJB occurs within a session. At the start of the session the SOAP client must invoke a "create" or "find" (in order to specify the bean object interest). While maintaining the same session, all other business methods are directed to that EJB. A subsequent "find" or "create" within the same or different session directs business method execution requests to the newly "created" (or "found") EJB.

Another issue is that EJB specification provides that some "find" methods can return either a Collection of EJB refs or single EJB ref.

The Oracle solution for Entity EJBs embraces the following solution for this problem:

It disallows find methods that return "Collections". This allows for the provider to uniquely specify an Entity EJB to target subsequent business method requests.

## Deployment and Use of the OracleAS SOAP EJB Provider

To install an EJB provider and deploy Web Services to the provider under OC4J, where the application server hosts both the SOAP servlet and the deployed EJB's, follow these steps:

1. Deploy an EJB provider to SOAP using a provider descriptor.

The provider descriptor specifies the following:

- EJB access credentials by the middle tier
- JNDI context factory class
- JNDI context factory URL
- Provider class name
- Provider id

2. Create the EJB Web Service:

- Define the associated EJB classes and package the EJB into an EAR file as defined by J2EE spec.



- Define the service descriptor which specifies following details of the EJB Web Service:
  - \* JNDI Location
  - \* Home interface class name
  - \* Application Deployment Name of this EJB Web Service in OC4J
  - \* The provider id to which this service is to be associated
- 3. Deploy ear file in OC4J. Modify the OC4J specific EJB descriptor to correct the JNDI location for the EJB (as described in sample README).

## Current Known EJB Provider Limitations

All service methods can only take primitive Java types as arguments to the methods. User-defined Java types are currently not supported.

## Using PL/SQL Stored Procedures With the SP Provider

The OracleAS SOAP Stored Procedure (SP) Provider supports exposing PL/SQL stored procedures or functions as SOAP services. The Oracle9i Database Server allows procedures implemented in other languages, including Java and C/C++, to be exposed using PL/SQL; these stored procedures are exposed as SOAP services through PL/SQL interfaces.

The SP Provider framework works by translating PL/SQL procedures into Java wrapper classes, and then exporting the generating Java classes as SOAP Java services.

## SP Provider Supported Functionality

The SP Provider supports the following:

- PL/SQL stored procedures. both procedures and functions (this document uses procedure to refer to both)
- IN parameter modes
- Packaged procedures only (top-level procedures must be wrapped in a package before they can be exported)
- Overloaded procedures (however, if two different PL/SQL types map to the same Java type during translating, there may be errors during the export of the PL/SQL package; these errors may be fixed by avoiding the overloading, or else by writing a new dummy package which does not contain the offending overloaded procedures)
- Simple types
- (user-defined) object types

## SP Provider Unsupported Functionality

The SP provider does not support the following:

- The SP Provider framework uses Oracle JPublisher to translate from PL/SQL to Java; hence, it inherits all of the restrictions of Oracle JPublisher.

## SP Provider Supported Simple PL/SQL Types

The SOAP SP provider supports the following simple types. NULL values are supported for all of the simple types listed, except NATURALN and POSITIVEN.

The Oracle JPublisher documentation provides full details on the mappings of these types.

- VARCHAR2 (STRING, VARCHAR)
- LONG
- CHAR (CHARACTER)
- NUMBER (DEC, DECIMAL, DOUBLE PRECISION, FLOAT, INTEGER, INT, NUMERIC, REAL, SMALLINT)
- PLS\_INTEGER
- BINARY\_INTEGER (NATURAL, NATURALN, POSITIVE, POSITIVEN)

## Using Object Types

Oracle JPublisher supports the use of user-defined object types. The SP Provider framework generates `oracle.sql.CustomDatum` style classes since these allow automatic serialization using the default `BeanSerializer` in SOAP.

Refer to the company sample for an example of using object types.

## Deploying a Stored Procedure Provider

[Example A-7](#) shows a sample provider deployment descriptor for a stored procedure. You may use any unique id for the provider name (the example uses "company-provider").

The attributes `user`, `password`, and `url` are used to create the URL to connect to the database, and they are all required. The number of connections for a service, handled by this provider, is set using `connections_per_service`; this is optional and defaults to 10.

Deploy the sample provider descriptor shown in [Example A-7](#), appropriately edited for the local configuration, using the provider manager.

### **Example A-7 Sample SP Provider Deployment Descriptor**

```
<isd:provider xmlns:isd="http://xmlns.oracle.com/soap/2001/04/deploy/provider"
  id="company-provider"
  class="oracle.soap.providers.sp.SpProvider">
  <!-- edit the following option "values" as appropriate -->
  <isd:option key="user" value="YOUR-USER-NAME" />
  <isd:option key="password" value="YOUR-PASSWORD" />
  <isd:option key="url" value="jdbc:oracle:thin:@YOUR-HOST:YOUR-PORT:YOUR-SID" />
  <isd:option key="connections_per_service" value="3" />
</isd:provider>
```

## Translating PL/SQL Stored Procedures into Java

The shell script `$SOAP_HOME/bin/sp2jar.sh` translates a PL/SQL package and all its contained procedures/functions into a Java class with equivalent methods. If the package uses any user-defined types, these types are also translated into equivalent Java classes.

The README file in the samples directory has an example of the usage of the `sp2jar.sh` command to translate the company example into a jar file of compiled Java classes. The README also describes how to load the PL/SQL packages into the database.

Let us assume for the rest of the document that a PL/SQL package company has been installed on a database, and it has been exported into a set of compiled Java classes available in the jar file `company.jar`.

The generated `company.jar` should be made available in the CLASSPATH of the SOAP servlet, just as for other Java services.

## Deploying a Stored Procedure Service

**Example A-8** shows a sample service deployment descriptor for a stored procedure. Notice that the `id` attribute in the provider element identifies the provider under which this service is deployed.

The service descriptor looks exactly like that for a Java service, since the SP Provider framework translated PL/SQL procedures into Java class methods. All of the information specific to PL/SQL are part of the provider descriptor---the service itself looks like a Java service.

If the procedures use object types, it is necessary to define a type mapping for each object type. The XML type name must be identical to the SQL type name and must be in UPPER CASE (see EMPLOYEE and ADDRESS below). The `javaType` attribute identifies the `oracle.sql.CustomDatum` type that was generated by Oracle JPublisher.

The default `BeanSerializer` can be used to serialize/deserialize the types.

The generated method names are in lower-case since this is the default setting of Oracle JPublisher.

Deploy the sample service descriptor shown in **Example A-8** using the service manager.

### **Example A-8 Sample Stored Procedure Service Deployment Descriptor**

```
<isd:service xmlns:isd="http://xmlns.oracle.com/soap/2001/04/deploy/service"
  id="urn:www-oracle-com:company"
  type="rpc" >

  <isd:provider
    id="company-provider"
    methods="addemp getemp getaddress getempinfo changesalary removeemp"
    scope="Application" >
    <isd:java class="samples.sp.company.Company"/>
  </isd:provider>

  <isd:mappings>
    <isd:map encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
      xmlns:x="urn:company-sample" qname="x:EMPLOYEE"
      javaType="samples.sp.company.Employee"
      java2XMLClassName="org.apache.soap.encoding.soapenc.BeanSerializer"
      xml2JavaClassName="org.apache.soap.encoding.soapenc.BeanSerializer"/>
    <isd:map encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
      xmlns:x="urn:company-sample" qname="x:ADDRESS"
      javaType="samples.sp.company.Address"
      java2XMLClassName="org.apache.soap.encoding.soapenc.BeanSerializer"
      xml2JavaClassName="org.apache.soap.encoding.soapenc.BeanSerializer"/>
  </isd:mappings>
```

```
<isd:faultListener class="org.apache.soap.server.DOMFaultListener"/>

</isd:service>
```

## Invoking a SOAP Service that is a Stored Procedure

SOAP services that are PL/SQL stored procedures are invoked in exactly the same manner as any other SOAP service. The `company.jar` file created during the translating/deployment of a PL/SQL package is also needed on the client-side to compile application programs that invoke the SOAP service (this jar file is needed only if the stored procedures have input/output types that are user-defined types; if the procedures use only builtin-types, the generated jar file is not needed on the client).

The `README` file in the `company` samples directory has instructions on how to compile and test the sample client.

## SOAP Troubleshooting and Limitations

This section lists several techniques for troubleshooting Oracle Application Server Web Services, including:

- [Tunneling Using the `TcpTunnelGui` Command](#)
- [Setting Configuration Options for Debugging](#)
- [Using DMS to Display Runtime Information](#)
- [SOAP Limitations for Java Type Precedence with Overloaded Methods](#)

## Tunneling Using the `TcpTunnelGui` Command

SOAP provides the `TcpTunnelGui` command to display messages sent between a SOAP client and a SOAP server. `TcpTunnelGui` listens on a TCP port, which is different than the SOAP server, and then forwards requests to the SOAP server.

Invoke `TcpTunnelGui` as follows:

```
java org.apache.soap.util.net.TcpTunnelGui TUNNEL-PORT SOAP-HOST SOAP-PORT
```

[Table A-7](#) lists the command line options for `TcpTunnelGui`.

**Table A-7** *TcpTunnelGui* Command Arguments

Argument	Description
TUNNEL-PORT	The port that <code>TcpTunnelGui</code> listens to on the same host as the client
SOAP-HOST	The host of the SOAP server
SOAP-PORT	The port of the SOAP server

For example, suppose the SOAP server is running as follows,

```
http://system1:8080/soap/servlet/soaprouter
```

You would then invoke `TcpTunnelGui` on port 8082 with this command:

```
java org.apache.soap.util.net.TcpTunnelGui 8082 system1 8080
```

To test a client and view the SOAP traffic, you would use the following SOAP URL in the client program:

```
http://system1:8082/soap/servlet/soaprouter
```

## Setting Configuration Options for Debugging

To add debugging information to the SOAP Request Handler Servlet log files, change the value of the `severity` option for in the file `soap.xml`. This file is placed in `soap.ear` file in the directory `$SOAP_HOME/lib` on UNIX or in `%SOAP_HOME%\lib` on Windows.

To modify the debugging option, expand the `soap.ear` file and modify the file `soap.xml` in the directory `webapps/soap/WEB-INF` on UNIX or in `webapps\soap\WEB-INF` on Windows, then redeploy the updated `soap.ear` file.

For example, the following `soap.xml` segment shows the value to set for `severity` to enable debugging:

```
<!-- severity can be: error, status, or debug -->
<osc:logger class="oracle.soap.server.impl.ServletLogger">
  <osc:option name="severity" value="debug" />
</osc:logger>
```

After stopping and restarting the SOAP Request Handler Servlet, you can view debug information in the file `x.log`. The file is in the directory `$ORACLE_HOME/Apache/logs` on UNIX or in `%ORACLE_HOME%\Apache\x\logs` on Windows.

## Using DMS to Display Runtime Information

Oracle Application Server Web Services is instrumented with DMS to gather information on the execution of the SOAP Request Handler Servlet, the Java Provider, and on individual services.

DMS information includes execution intervals from start to stop for the following:

- Total time spent in SOAP request and response (includes time in providers and services)
- Total time spent in the Java Provider (includes time in services)
- Total time executing services (`soap/java-provider/service-URI`)

To view the DMS information, go to the following site:

```
http://hostname:port/soap/servlet/Spy
```

## SOAP Limitations for Java Type Precedence with Overloaded Methods

OracleAS SOAP supports Java inbuilt (primitive) types, wrapper types, one dimensional arrays of inbuilt types, and one dimensional arrays of wrapper types as parameters for SOAP RPC.

An inbuilt type parameter always takes precedence to a wrapper type parameter when the Java provider searches for an overloaded method. When there isn't a clear winner, for an overloaded method, a fault with appropriate message is returned.

For example:

A java class containing `aMethod(int)` hides `aMethod(Integer)` in the same class.

A java class containing `aMethod(int[])` hides `aMethod(Integer[])` in the same class.

A java class, when deployed as a SOAP RPC service returns a fault when a client invokes `aMethod()` containing the signatures, `aMethod(int, Float)` and `aMethod(Integer, float)`. In this case, there is no clear winner for resolving the precedence of the overloaded `aMethod()`.

## OracleAS SOAP Differences From Apache SOAP

This section covers differences between Apache Soap and OracleAS SOAP.

### Service Installation Differences

Additional instructions are provided for installing services when OracleAS SOAP is used in conjunction with OC4J.

### Optional Provider Enhancements

OracleAS SOAP supports both the Apache Provider interface, defined in `org.apache.soap.util.Provider`, and an enhanced provider interface, defined in `oracle.soap.server.Provider`.

The native Apache provider includes only two methods, `locate()` and `invoke()`. The Oracle Provider interface combines the `locate` and `invoke` methods, so that the provider does not have to store input parameters between the `locate()` and `invoke()` calls. Additionally, the Oracle Provider interface has `init()` and `destroy()` methods, which the SOAP servlet calls only once when the provider is instantiated. This allows providers to perform one time initialization such as opening a database or network connection, and to perform one time clean up activities.

When using the Apache provider interface, a single deployment descriptor supplies both service and provider properties. When using the Oracle Provider interface, these properties are separated between a service descriptor file and a provider descriptor file. This allows common provider properties to be shared among services. When a provider property changes, only a single descriptor file must be changed. Please see the Deployment section of this document for more information.

### Oracle Transport libraries

Oracle transport libraries are included for use with SOAP clients. Use of these libraries enables use of the Oracle Wallet Manager for keeping certificates securely, and use of the `HttpClient` libraries for HTTP connection management. The `HttpClient` libraries fix a security problem in the native Apache code which incorrectly returns cookies to servers other than the originating server.

### Modifications to Apache EJB Provider

The Apache EJB provider has been modified to work with the OC4J EJB container. In addition, the client interface to services provided by stateful and entity EJB's has been improved. The EJB handle is contained in the `HttpSession` association with the connection rather than being concatenated to the returned URL. Since the `HttpSession` cookie is handled transparently by the SOAP client, no special coding is required in the client.

## Stored Procedure Provider

A special provider has been added which allows services to be written using PL/SQL Stored Procedures or Functions.

## Utility Enhancements

The `wsdl2java` and `java2wsdl` scripts simplify building client side code from WSDL descriptions and for generating WSDL descriptions of Java services.

## Modifications to Sample Code

The Apache samples have been modified to work with OracleAS SOAP and OC4J. The `com`, `calculator`, `weblogic_ejb` samples have been omitted. New samples illustrating use of Oracle Stored Procedures and OC4J EJB's as Web Services have been added.

## Handling the `mustUnderstand` Attribute in the SOAP Header

This section describes the check that is performed for the `mustUnderstand` attribute within the header blocks of the SOAP envelope, and describes the difference between the Apache SOAP and the OracleAS SOAP processing of this attribute.

### Setting the `mustUnderstand` Check

The check for the `mustUnderstand` attribute is enabled in the deployment descriptor of the service by setting the `checkMustUnderstands` flag. If this flag set to `true`, the check for the `mustUnderstand` attribute within each header block is performed. If the `checkMustUnderstands` flag is set to `false`, the check for the `mustUnderstand` attribute is not performed. The default value of `checkMustUnderstands` flag is `true`.

### How the `mustUnderstand` Check Works

If the `checkMustUnderstands` flag is set to `true`, then a check is made on all header entries of the envelope after the global request handlers have finished processing and before handing the envelope to the appropriate service. At this point, if any header entries contain a `mustUnderstand` attribute that is set to `true` or to "1", then an exception is thrown. Note, the global handler(s) can be used to process one or more header blocks that have the `mustUnderstand` attribute set to `true`.

If the `checkMustUnderstands` flag is set to `false`, then header entries of the envelope are not checked to see if any entries contain a `mustUnderstand` attribute that is set to `true` or to "1". It is then understood that it is up to the service implementation to make sure that this check is done before processing the body of the envelope.

### Differences Between Apache SOAP and Oracle SOAP for `mustUnderstand`

The differences between Apache SOAP and OracleAS SOAP with respect to the handling of the `mustUnderstand` attribute are the following:

1. In the Apache service deployment descriptor and the Oracle Service deployment descriptor, you may include the `checkMustUnderstands` attribute. In Apache, the default value of the `checkMustUnderstands` attribute is `false`, in OracleAS SOAP the default value of this attribute is `true`.
2. In Apache SOAP, if the service deployment descriptor contains `checkMustUnderstands='true'` and a message with `mustUnderstand='1'`

or `mustUnderstand="true"` arrives at the server then a fault is sent back with the fault code value of:

`mustUnderstand`

This fault code is not namespace qualified and is incorrect.

In OracleAS SOAP the fault code that is sent back is namespace qualified and is defined by SOAP 1.1:

`SOAP-ENV:MustUnderstand`

3. In Apache SOAP, the `mustUnderstand` attribute has to be handled by the service implementation. In OracleAS SOAP, the `mustUnderstand` attribute can be either handled in the SOAP handlers or in the service implementation. This is very useful for processing headers (with `mustUnderstand` set to '1') which have a 'global' use. Examples of such headers/functionality are encryption, `digest`, authentication, logging etc.

## Apache Software License, Version 1.1

This program contains third-party code from the Apache Software Foundation (Apache). Under the terms of the Apache license, Oracle is required to provide the following notices. Note, however, that the Oracle program license that accompanied this product determines your right to use the Oracle program, including the Apache software, and the terms contained in the following notices do not change those rights. Notwithstanding anything to the contrary in the Oracle program license, the Apache software is provided by Oracle AS IS and without warranty or support of any kind from Oracle or Apache.

=====

The Apache Software License, Version 1.1

Copyright (c) 2000 The Apache Software Foundation. All rights reserved.

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

1. Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.
2. Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.
3. The end-user documentation included with the redistribution, if any, must include the following acknowledgment:

"This product includes software developed by the Apache Software Foundation (<http://www.apache.org/>)."

Alternately, this acknowledgment may appear in the software itself, if and wherever such third-party acknowledgments normally appear.

4. The names "Apache" and "Apache Software Foundation" must not be used to endorse or promote products derived from this software without prior written permission. For written permission, please contact [apache@apache.org](mailto:apache@apache.org).
5. Products derived from this software may not be called "Apache", nor may "Apache" appear in their name, without prior written permission of the Apache Software Foundation.



THIS SOFTWARE IS PROVIDED ``AS IS" AND ANY EXPRESSED OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE APACHE SOFTWARE FOUNDATION OR ITS CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

This software consists of voluntary contributions made by many individuals on behalf of the Apache Software Foundation. For more information on the Apache Software Foundation, please see <http://www.apache.org/>.

Portions of this software are based upon public domain software originally written at the National Center for Supercomputing Applications, University of Illinois, Urbana-Champaign.



---

---

## Web Services Security

The ability to control user access to Web content and to protect your site against people breaking into your system is critical. This appendix describes the architecture and configuration of security for Oracle Application Server Web Services, including the Oracle Application Server UDDI Registry.

This chapter covers the following topics:

- [About Web Services Security](#)
- [Configuring Web Services Security](#)
- [About Oracle Application Server UDDI Registry Security](#)
- [Configuring UDDI Security](#)

**See Also:**

- *Oracle Application Server 10g Security Guide*
- *Oracle Identity Management Concepts and Deployment Planning Guide*

## About Web Services Security

SOAP is the messaging protocol for Oracle Application Server Web Services. Oracle Application Server Web Services only supports HTTP (S) for a transport protocol for SOAP messages. Oracle Application Server security that applies for HTTP(S) can be leveraged for Oracle Application Server Web Services.

Oracle Application Server Web Services supports the following security features:

- **Secure Connection:** By securing the connection using SSL (HTTPS), one can invoke a Web Service securely.
- **Authentication:** Basic and Digest Access Authentication can be enforced using HTTP (S) headers. This method is not secure unless the authentication is specified in conjunction with SSL.
- **Authorization:** Authorization is supported by retrieving the Principal using a User Manager such as the Oracle Application Server Java Authentication and Authorization Service (JAZN) User Manager.

All the HTTP(S) transport security features are applicable to all types of Oracle Application Server Web Services implementations (including stateless and stateful java classes, stateless session bean and stateless stored procedures). In addition, if a stateless session bean is exposed as a Web Service, ACL policies can be enforced on the bean when the connection is authorized by a User Manager and a Principal object is obtained.

If a stored procedure is exposed as a Web Service, then it is secure to encrypt the password of the corresponding data source in the data-sources.xml file.

**See Also:**

- *Oracle Application Server Containers for J2EE Security Guide*
- Chapter 8, "Configuring EJB Application Security" in the *Oracle Application Server Containers for J2EE Enterprise JavaBeans Developer's Guide*

## Configuring Web Services Security

When you run a client-side application that uses Oracle Application Server Web Services, you can access secure Web Services by setting properties in the client application. [Table B-1](#) shows the available properties that provide credentials and other security information for Web Services clients.

In a Web Services client application, you can set the security properties shown in [Table B-1](#) as system properties by using the `-D` flag at the Java command line, or you can also set security properties in the Java program by adding these properties to the system properties (use `System.setProperty()` to add properties). In addition, the client side stubs include the `_setTransportProperties` method that is a public method in the client proxy stubs. This method enables you to set the appropriate values for security properties by supplying a `Properties` argument.

**Table B-1 Web Services HTTP Transport Security Properties**

Property	Description
<code>http.authRealm</code>	Specifies the realm for which the HTTP authentication username/password is specified.  This property is mandatory when using basic authentication.
<code>http.authType</code>	Specifies the HTTP authentication type. The case of the value specified is ignored.  Valid values: <code>basic</code> , <code>digest</code>  The value <code>basic</code> specifies HTTP basic authentication.  Specifying any value other than <code>basic</code> or <code>digest</code> is the same as not setting the property.
<code>http.password</code>	Specifies the HTTP authentication password.
<code>http.proxyAuthRealm</code>	Specifies the realm for which the proxy authentication username/password is specified.
<code>http.proxyAuthType</code>	Specifies the proxy authentication type. The case of the value specified is ignored.  Valid values: <code>basic</code> , <code>digest</code>  Specifying any value other than <code>basic</code> or <code>digest</code> is the same as not setting the property.
<code>http.proxyHost</code>	Specifies the hostname or IP address of the proxy host.
<code>http.proxyPassword</code>	Specifies the HTTP proxy authentication password.
<code>http.proxyPort</code>	Specifies the proxy port. The specified value must be an integer. This property is only used when <code>http.proxyHost</code> is defined; otherwise this value is ignored.  Default value: 80
<code>http.proxyUsername</code>	Specifies the HTTP proxy authentication username.
<code>http.username</code>	Specifies the HTTP authentication username.
<code>java.protocol.handler.pkgs</code>	Specifies a list of package prefixes for <code>java.net.URLStreamHandlerFactory</code> . The prefixes should be separated by " " vertical bar characters.  This value should contain: <code>HTTPClient</code>  This value is required by the Java protocol handler framework; it is not defined by Oracle Application Server. This property must be set when using HTTPS. If this property is not set using HTTPS, a <code>java.net.MalformedURLException</code> is thrown.  <b>Note:</b> This property must be set as a system property.  For example, set this property as shown in either of the following: <ul style="list-style-type: none"> <li>■ <code>java.protocol.handler.pkgs=HTTPClient</code></li> <li>■ <code>java.protocol.handler.pkgs=sun.net.www.protocol   HTTPClient</code></li> </ul>

**Table B-1 (Cont.) Web Services HTTP Transport Security Properties**

Property	Description
<code>oracle.soap.transport.allowUserInteraction</code>	<p>Specifies the allows user interaction parameter. The case of the value specified is ignored. When this property is set to <code>true</code> and either of the following are true, the user is prompted for a username and password:</p> <ol style="list-style-type: none"> <li>1. If any of properties <code>http.authType</code>, <code>http.username</code>, or <code>http.password</code> is not set, and a 401 HTTP status is returned by the HTTP server.</li> <li>2. If either of properties <code>http.proxyAuthType</code>, <code>http.proxyUsername</code>, or <code>http.proxyPassword</code> is not set and a 407 HTTP response is returned by the HTTP proxy.</li> </ol> <p>Valid values: <code>true</code>, <code>false</code></p> <p>Specifying any value other than <code>true</code> is considered as <code>false</code>.</p>
<code>oracle.ssl.ciphers</code>	<p>Specifies a list of: separated cipher suites that are enabled.</p> <p>Default value: The list of all cipher suites supported with Oracle SSL.</p>
<code>oracle.wallet.location</code>	<p>Specifies the location of an exported Oracle wallet or exported trustpoints.</p> <p>Note: The value used is not a URL but a file location, for example:</p> <p><code>/etc/ORACLE/Wallets/system1/exported_wallet</code> (on UNIX)</p> <p><code>d:\oracle\system1\exported_wallet</code> (on Windows)</p> <p>This property must be set when HTTPS is used with SSL authentication, server or mutual, as the transport.</p>
<code>oracle.wallet.password</code>	<p>Specifies the password of an exported wallet. Setting this property is required when HTTPS is used with client, mutual authentication as the transport.</p>

## About Oracle Application Server UDDI Registry Security

This section covers the following topics:

- [Protecting Oracle Application Server UDDI Registry Resources](#)
- [Managing and Enforcing Protected UDDI Resources](#)
- [Using Oracle Application Server Security Services](#)

**See Also:** ["OracleAS UDDI Registry Administration"](#) on page 10-34

## Protecting Oracle Application Server UDDI Registry Resources

Oracle Application Server UDDI resources are protected as follows.

### Oracle Application Server UDDI Registry

For the OracleAS UDDI Registry, the following resources are protected:

- Data – Write access to the data stored in the OracleAS UDDI Registry is protected; this is typically metadata of Web Services.
- Functions – Administrative operations to the OracleAS UDDI Registry.
- Passwords – N/A. User passwords are protected by JAZN.

### Oracle Application Server Content Subscription Manager Application

For the Oracle Application Server UDDI Content Subscription Manager application, the following resource is protected:

- Passwords – Password for the UDDI syndication subscriber are protected.

## Managing and Enforcing Protected UDDI Resources

Protection for the following OracleAS UDDI Registry resources are managed and enforced as follows.

### Oracle Application Server UDDI Registry

Oracle Application Server Java Authentication and Authorization Service (JAZN) and the UDDI application manages and enforces write access to the data stored in the OracleAS UDDI Registry. JAZN determines the identity and the security role of a user. Only the owner has rights to update data.

For administrative operations for the OracleAS UDDI Registry JAZN also manages and enforces access; in addition, JAZN protects the servlets that provide administrative operations.

### Oracle Application Server Content Subscription Manager Application

The application manages the UDDI syndication subscription password used to access Oracle Application Server Syndication Services. The password, which is persistently stored in the database, is further protected by the database `DBMS_OBFUSCATION` PL/SQL package.

Update of the UDDI syndication subscriber password is available through a UDDI Web-based tool. The web-based tool uses JAZN to query the security role of the authenticated user. The password update facility is available only if the authenticated user has the `uddiadmin` security role.

## Using Oracle Application Server Security Services

UDDI leverages the JAZN User level security features and uses SSL encryption, both server side and client side, for accessing OracleAS Infrastructure 10g options.

## Configuring UDDI Security

To configure UDDI for security, consider the following areas:

- [Configuring the Oracle Application Server UDDI Registry](#)
- [Configuring the UDDI Content Subscription Manager](#)
- [Configuring the UDDI Client](#)

### Configuring the Oracle Application Server UDDI Registry

To ensure the confidentiality of the communication between the OracleAS UDDI Registry and clients, do the following:

1. Configure the Oracle HTTP Server/SSL listener to provide HTTPS access.
2. Configure OC4J to prohibit HTTP access.

3. To ensure the communication to a UDDI replication endpoint is authorized, configure the Oracle HTTP Server/SSL listener to enable HTTPS client-certificate based authentication.

Configure all security-sensitive UDDI endpoints, including: publishing, administration, replication wallet administration, and subscription management (typically, the inquiry endpoint does not need to be confidential).

### **Configuring the UDDI Content Subscription Manager**

In order to make the Oracle Application Server Content Subscription Manager functional, you must supply the proper password of the UDDI syndication subscriber.

### **Configuring the UDDI Client**

If you use the UDDI Client Library to develop applications to communicate with the OracleAS UDDI Registry, you can use the Oracle Application Server Web Services security features to configure the HTTP transport properties.

**See Also:** ["Configuring Web Services Security"](#) on page B-2



---

---

# Troubleshooting OracleAS Web Services

This appendix describes common problems that you might encounter when using Oracle Application Server Web Services and explains how to solve them. It contains the following topics:

- [Appendix C.1, "Problems and Solutions"](#)
- [Appendix C.2, "Diagnosing OracleAS Web Services Problems"](#)
- [Appendix C.3, "Need More Help?"](#)

## C.1 Problems and Solutions

This section describes common problems and solutions. It contains the following topics:

- [Appendix C.1.1, "Receiving "Unsupported Response Content Type" Error"](#)
- [Appendix C.1.2, "Cannot Publish Doc/Literal in JDeveloper"](#)
- [Appendix C.1.3, "Cannot Register Web Service"](#)
- [Appendix C.1.4, "UDDI Registry Screens Missing"](#)
- [Appendix C.1.5, "UDDI Management Screens Are Not Enabled"](#)

### C.1.1 Receiving "Unsupported Response Content Type" Error

Invoking a Web service produces an "unsupported response content type" error.

#### **Problem**

A likely cause of the error is that the URI address of the endpoint in the stub or dynamic proxy is incorrect. With an incorrect URI address, the client receives a native HTTP response without a SOAP payload which results in an "unsupported response content type" error.

#### **Solution**

Ensure that the URI address is correct.

### C.1.2 Cannot Publish Doc/Literal in JDeveloper

Web service cannot publish doc/literal in JDeveloper 9.0.3/9.0.4/9.0.5.1.

**Problem**

Trying to publish doc/literals without using Stateless Java Classes and JMS destinations.

**Solution**

Since Oracle Application Server 9.0.3, Oracle Application Server Web Services has supported doc/literal operations using Stateless Java Classes and JMS destinations. Information for this support is available in the Oracle Application Server documentation at:

[http://download-west.oracle.com/docs/cd/B10464\\_02/web.904/b10447/docservices.htm#sthref259](http://download-west.oracle.com/docs/cd/B10464_02/web.904/b10447/docservices.htm#sthref259)

### C.1.3 Cannot Register Web Service

When deploying a Web service, using Oracle Application Server Control, the Web service cannot be registered in the Oracle Application Server UDDI Registry.

**Problem**

Although the button to register the service appears, it is not enabled. This problem often indicates that you have not installed the Oracle Application Server Portal. The Oracle Application Server UDDI Server is a J2EE servlet application that is deployed automatically by the Oracle Application Server Installer into the same J2EE OC4J container as the Oracle Application Server Portal.

**Solution**

Install the Oracle Application Server Portal. After installation, you must ping the servlet at least once to initialize the registry and enable the Application Server Control UDDI user interface.

### C.1.4 UDDI Registry Screens Missing

The Oracle Application Server UDDI Registry screens in the Application Server Control are missing.

**Problem**

This problem indicates that the Oracle Application Server Portal is not installed. Requiring that the Oracle Application Server Portal be installed, before the Oracle Application Server UDDI Registry screens become available, ensures that customers have the Oracle Application Server MetaData Repository installed, which is a pre-requisite for Oracle Application Server UDDI management.

**Solution**

Install the Oracle Application Server Portal. After installation, you must ping the servlet at least once to initialize the registry and enable the Application Server Control UDDI user interface.

### C.1.5 UDDI Management Screens Are Not Enabled

The Oracle Application Server Control UDDI management screens are not enabled.

---

### Problem

The Oracle Application Server Control UDDI management screens are not enabled. The Oracle Application UDDI Registry is installed on another instance of Oracle Application Server.

### Solution

The Oracle Application UDDI Registry must be installed in the same instance of Oracle Application Server as the Oracle Application Server Control UDDI management screens for the screens to be enabled.

## C.2 Diagnosing OracleAS Web Services Problems

Oracle Application Server can consist of many components. To aid in diagnosing problems specific to Web Services, you can configure Oracle Application Server to generate diagnostic messages specific to Web Services. The following section provides information on how to configure Oracle Application Server to generate diagnostic messages for Web Services.

- [Appendix C.2.1, "Generating Web Services Diagnostic Messages"](#)

### C.2.1 Generating Web Services Diagnostic Messages

To obtain diagnostic information specific to Web services, set the system property `ws.debug` to `True`.

- To set `ws.debug` in an OC4J standalone version, at the command line enter:

```
java Dws.debug=true ... -jar oc4j.jar
```

- For the OC4J Java, Standard, or Enterprise editions, `ws.debug` can be set from within the Oracle Application Server Control. From the Administration tab of the OC4J instance where the Web service is deployed, navigate to the Server properties link. In the Java options text field, add:

```
Dws.debug=true
```

The diagnostic messages are located in the redirected output/errors log. You can find the log in the Log Viewer Application Server Control screens.

## C.3 Need More Help?

You can find more solutions on Oracle *MetaLink*, <http://metalink.oracle.com>. If you do not find a solution for your problem, log a service request.

### See Also:

- *Oracle Application Server Release Notes*, available on the Oracle Technology Network:  
<http://www.oracle.com/technology/documentation/index.html>



---

---

# Glossary

## Dynamic Web Service Client

When you want to use Web Services, you can develop a **dynamic Web Service client**. With A dynamic client the client performs a lookup to find the Web Service's location in a OracleAS UDDI Registry before accessing the service.

## SOAP

SOAP is the name of a lightweight, XML-based protocol for exchanging information in a decentralized, distributed environment. SOAP supports different styles of information exchange, including: Remote Procedure Call style (RPC) and Message-oriented exchange.

**See Also:** <http://www.w3.org/TR/SOAP/> for information on SOAP 1.1 specification

## Static Web Service Client

When you want to use Web Services, you can develop a **static client**. A static client knows where a Web Service is located without looking up the service in a OracleAS UDDI Registry.

## Stored Procedure Web Service

Oracle Application Server Web Services implemented as stateless PL/SQL Stored Procedures or Functions are called **Stored Procedure Web Services**. Stored Procedure Web Services enable you to export, as services running under Oracle Application Server Web Services, PL/SQL procedures and functions that run on an Oracle database server.

## UDDI

Universal Description, Discovery, and Integration (UDDI) is a specification for an online electronic registry that serves as electronic *Yellow Pages*, providing an information structure where various business entities register themselves and the services they offer through their WSDL definitions.

**See Also:** <http://www.uddi.org> for information on Universal Description, Discovery and Integration specifications.

## Web Service

A Web Service is a discrete business process that does the following:

- Exposes and describes itself – A Web Service defines its functionality and attributes so that other applications can understand it. A Web Service makes this functionality available to other applications.

- 
- Allows other services to locate it on the web – A Web Service can be registered in an electronic *Yellow Pages*, so that applications can easily locate it.
  - Can be invoked – Once a Web Service has been located and examined, the remote application can invoke the service using an Internet standard protocol.
  - Returns a response – When a Web Service is invoked, the results are passed back to the requesting application over the same Internet standard protocol that is used to invoke the service.

### **Web Services Description Language (WSDL)**

Web Services Description Language (WSDL) is an XML format for describing network services containing RPC-oriented and message-oriented information. Programmers or automated development tools can create WSDL files to describe a service and can make the description available over the Internet.

**See Also:** <http://www.w3.org/TR/wsdl> for information on the Web Services Description Language (WSDL) format.

## A

---

- accept-untyped-request configuration tag, 12-3
- access point
  - creating with
    - UDDI client API, 10-31
    - UDDI publishing tool, 10-26
- addPublisherAssertion method
  - UDDI Client API, 10-33
- addressTModelKeyValidation property, 10-69
- addUddiElement method
  - UDDI Client API, 10-12, 10-30, 10-31
- administration end point
  - for UDDI, 10-7
- assertionKeyedRefValidation property, 10-46, 10-69
- AssertionStatusReport instance
  - UDDI Client API, 10-33
- authenticate method
  - UDDI Client API, 10-29
- authentication
  - UDDI publishing and, 10-29

## B

---

- bindingTemplate data structures, 10-2
  - creating with
    - publishing tool, 10-26
    - UDDI client API, 10-31
- businessEntity data structures, 10-2
  - creating with
    - UDDI Client API, 10-32
    - UDDI publishing tool, 10-22
  - publishing, 10-32
  - setting prefix, 10-7
- businessEntityURLPrefix property, 10-7, 10-69
- businessService data structures, 10-2
  - creating with
    - UDDI client API, 10-31
    - UDDI publishing tool, 10-25

## C

---

- categories
  - for UDDI registry, 10-4
- CategoryBag data structures
  - UDDI Client API, 10-30, 10-31

- CategoryBag instance
  - UDDI Client API, 10-11
- categoryValidation property, 10-46, 10-70
- categoryValidationTModelKeys property, 10-46, 10-70
- changeOwner option, 10-40, 10-62
- changeRecordWantsAck property, 10-43, 10-71
- classification support for UDDI, 10-4, 10-5
- classifications
  - for UDDI, 10-4
- class-name configuration tag, 3-9, 6-8
- client-side certification
  - UDDI replication and, 10-42
- client-side proxies
  - generating for Web Services, 8-5
  - getting directly, 8-3
  - using, 8-7
- client-side request header support, 12-3
- close method
  - UDDI Client API, 10-13
- configuration
  - UDDI registry, 10-7, 10-8
- connection-factory-resource-ref configuration tag, 7-10
- consuming Web Services
  - SOAP-based Web services using WSDL document, 11-1
  - running the demonstration, 11-7
  - using configuration file, 11-3
  - using wsdl2ejb utility command-line options, 11-1
- context configuration tag, 3-7
- correctChangeRecord option, 10-43, 10-62
- createAccessPoint method
  - UDDI Client API, 10-31
- createBindingTemplates method
  - UDDI Client API, 10-31
- createBusinessEntity method
  - UDDI Client API, 10-32
- createBusinessServices method
  - UDDI Client API, 10-31
- createCategoryBag method
  - UDDI Client API, 10-30, 10-31
- createFindQualifiers method
  - UDDI Client API, 10-12
- createIdentityPublisherAssertion method

- UDDI Client API, 10-34
- createKeyedReference method
  - UDDI Client API, 10-30
- createOverviewDoc method
  - UDDI Client API, 10-30
- createTModel method
  - UDDI Client API, 10-30
- createTModelInstanceDetails method
  - UDDI Client API, 10-31
- createUddiClient method
  - UDDI Client API, 10-32
- createWriterXmlWriter method
  - UDDI Client API, 10-32
- custody transfers
  - UDDI entities, 10-43

## D

---

- database-JNDI-name configuration tag, 5-5
- data-sources.xml configuration file, 5-8
- db-pkg-name configuration tag, 5-5
- db-url configuration tag, 5-5
- debugging using ws.debug property, 12-1
- default language
  - for UDDI registry, 10-8, 10-71
- defaultLang property, 10-8, 10-71
- deleteEntity option, 10-40, 10-62
- deletePublisherAssertion method
  - UDDI Client API, 10-34
- deleteRoleQuotaLimits option, 10-39, 10-62
- description configuration tag, 3-7
- destination-path configuration tag, 3-7
- destroyTModel option, 10-40, 10-63
- discovering Web Services, 10-4, 10-9
- Discovery URL
  - UDDI and, 10-24
- display-name configuration tag, 3-7
- document style interface, 6-5
- doPing option, 10-44, 10-63
- downloadReplicationConfiguration option, 10-42, 10-63
- Dun & Bradstreet D-U-N-S Number identifier system, 10-5
- D-U-N-S, 10-5
- dynamic invocation of Web Services, 11-16
  - dynamic invocation API, 11-16
  - WebServiceProxy client, 11-19

## E

---

- ejb-name configuration tag, 4-8
- ejb-resource configuration tag, 3-9, 4-8
- EJBs sample code, 4-1
- Element
  - arrays of, 3-5, 4-5, 6-2
  - null values, 6-2
- entities
  - changing authorized name, 10-41
  - changing operator name, 10-41
  - changing ownership, 10-40, 10-62

- changing timestamp, 10-41
- deleting, 10-40, 10-62
  - setting operational information, 10-41
- error handling
  - UDDI replication, 10-43
- exception handling
  - UDDI replication, 10-43
- external validation, 10-47, 10-48
  - UDDI registry, 10-47
- externalValidation property, 10-47, 10-72
- externalValidationTimeout property, 10-48, 10-72
- externalValidationTModelList property, 10-48, 10-72

## F

---

- findBusiness method
  - UDDI Client API, 10-11
- findService method
  - UDDI Client API, 10-12

## G

---

- generating client proxies, A-6
- generating WSDL documents, A-7
- getAssertionStatusReport method
  - UDDI Client API, 10-33
- getChangeRecord option, 10-44, 10-63
- getHighWaterMarks option, 10-44, 10-64
- getProperties option, 10-35, 10-64
- getRoleQuotaLimits option, 10-39, 10-64
- getting client-side proxies for Web Services, 8-2
- getting WSDL descriptions for Web Services, 8-2
- getUddiElementFactory method
  - UDDI Client API, 10-29
- getUserDetail option, 10-37, 10-64
- getUsers option, 10-65
  - UDDI registry, 10-37
- groups
  - UDDI registry, 10-36

## H

---

- header support, 12-3
- hostingRedirectorValidation property, 10-46, 10-73
- http
  - //metalink.oracle.com, C-3
- HTTP transport properties
  - http.authRealm property, 8-9, B-3
  - http.authType property, 8-9, B-3
  - http.password property, 8-9, B-3
  - http.proxyAuthRealm property, 8-9, B-3
  - http.proxyAuthType property, 8-9, B-3
  - http.proxyHost property, 8-9, B-3
  - http.proxyPassword property, 8-9, B-3
  - http.proxyPort property, 8-9, B-3
  - http.proxyUsername property, 8-9, B-3
  - http.username property, 8-9, B-3
  - java.protocol.handler.pkgs property, 8-10, B-3
  - oracle.wallet.location property, 8-10, B-4
  - oracle.soap.transport.allowUserInteraction property, 8-10, B-4



- oracle.ssl.ciphers property, 8-10, B-4
- oracle.wallet.password property, 8-10, B-4
- http.authRealm property, 8-9, B-3
- http.authType property, 8-9, B-3
- http.password property, 8-9, B-3
- http.proxyAuthRealm property, 8-9, B-3
- http.proxyAuthType property, 8-9, B-3
- http.proxyHost property, 8-9, B-3
- http.proxyPassword property, 8-9, B-3
- http.proxyPort property, 8-9, B-3
- http.proxyUsername property, 8-9, B-3
- HTTPS
  - client-side certification
    - UDDI replication and, 10-42
- http.username property, 8-9, B-3

## I

---

- identifier support for UDDI, 10-5
- identifier taxonomies, 10-5
- identifiers
  - for UDDI, 10-21
  - UUID, 10-5
- identifierValidation property, 10-46, 10-73
- import option, 10-65
  - UDDI registry, 10-40
- importing entities, 10-40
- initialization
  - UDDI registry, 10-6
- inquiry API
  - for UDDI registry, 10-4, 10-10
- inquiry SOAP end point, 10-7
- interface-name configuration tag, 3-9, 6-8
- interoperability with .NET, 12-1
- ISO-3166 Geographic classification (ISO-3166), 10-4
- ISO-3166 taxonomy, 10-4

## J

---

- jar-generation configuration tag, 5-5
- Java Beans, 3-5, 4-4
- java class interface, 3-4
- java2wsdl script, A-7
- java.protocol.handler.pkgs property, 8-10, B-3
- java-resource configuration tag, 3-9, 6-8
- jdbcDriverType property, 10-74
- jms-delivery-mode configuration tag, 7-10
- jms-doc-service configuration tag, 7-10
- jms-expiration configuration tag, 7-10
- jms-message-type configuration tag, 7-10
- jms-priority configuration tag, 7-11

## L

---

- language
  - default for UDDI registry, 10-8, 10-71
- locating Web Services, 8-1

## M

---

- maxChangeRecordsSentEachTime property, 10-44,

- 10-74
- maxConnections property, 10-74
- message-style configuration tag, 3-9, 6-9
- method-name configuration tag, 5-5
- minConnections property, 10-75

## N

---

- NAICS taxonomy, 10-4
- .NET interoperability, 12-1
- North American Industry Classification System (NAICS), 10-4

## O

---

- OC4J startup using ws.debug, 12-1
- operation configuration tag, 7-11
- operational information
  - setting for entities, 10-41
- operator name
  - changing for entity, 10-41
- operatorCategory property, 10-46, 10-75
- operatorName property, 10-7, 10-76
- option name="force" configuration tag, 9-4
- option name="httpServerURL" configuration tag, 9-4
- option name="include-source" configuration tag, 8-6
- option name="packageIt" configuration tag, 9-4
- option name="source-path" configuration tag, 3-7
- option name="wsdl-location" configuration tag, 8-6
- option package-name configuration tag, 8-6
- Oracle SOAP, A-16
  - audit logger
    - configuring, A-13
    - filter, A-12
    - HostName, A-11
    - IpAddress, A-11
    - Method element, A-11
    - schema, A-11
    - ServiceURI element, A-11
    - TimeStamp element, A-11
    - User element, A-11
  - auditLogDirectory option, A-14
  - client API
    - security features, A-15
  - configuration
    - handlers, A-10
    - soap.xml, A-2
  - debugging
    - setting values in soap.xml, A-27
  - deploying services, A-5
  - deployment descriptor, A-8
  - error handlers, A-10
  - errorHandlers deployment parameter, A-3
  - faultListeners deployment parameter, A-3
  - filter option, A-14
  - handlers
    - deployment parameter, A-3
    - error, A-10
    - request, A-10

- response, A-10
- HostName element, A-11
- HTTP transport properties
  - http.authRealm property, A-16
  - http.authType property, A-16
  - http.password property, A-16
  - http.proxyAuthRealm property, A-16
  - http.proxyAuthType property, A-16
  - http.proxyHost property, A-16
  - http.proxyPassword property, A-16
  - http.proxyPort property, A-17
  - http.proxyUsername property, A-17
  - http.username property, A-17
- java.protocol.handler.pkgs property, A-17
- oracle.wallet.location property, A-18
- oracle.soap.transport.allowUserInteraction property, A-17
- oracle.wallet.password property, A-18
- http.authRealm property, A-16
- http.authType property, A-16
- http.password property, A-16
- http.proxyAuthRealm property, A-16
- http.proxyAuthType property, A-16
- http.proxyHost property, A-16
- http.proxyPassword property, A-16
- http.proxyPort property, A-17
- http.proxyUsername property, A-17
- http.username property, A-17
- includeRequest option, A-14
- includeResponse option, A-14
- IpAddress element, A-11
- java.protocol.handler.pkgs property, A-17
- listing services, A-6
- logger
  - setting values in soap.xml, A-27
- logger deployment parameter, A-3
- Method element, A-11
- oracle.wallet.location property, A-18
- oracle.soap.transport.allowUserInteraction property, A-17
- oracle.soap.transport.1022ContentType property, A-17
- oracle.ssl.ciphers property, A-18
- oracle.wallet.password property, A-18
- providerManager deployment parameter, A-4
- querying services, A-6
- request handlers, A-10
- requestHandlers deployment parameter, A-4
- response handlers, A-10
- responseHandlers deployment parameter, A-4
- security features, A-15
- service manager
  - deploying services, A-5
  - listing services, A-6
  - querying services, A-6
  - undeploying services, A-5
  - verifying services, A-6
- serviceManager deployment parameter, A-4
- ServiceURI element, A-11
- servlet.soaprouter.initArgs parameter, A-2

- soap.properties
  - soapConfig, A-2
- soap.xml, A-2
- TcpTunnelGui command, A-26
- TimeStamp element, A-11
- troubleshooting, A-26
- undeploying services, A-5
- User element, A-11
- oracle.wallet.location property, 8-10, B-4
- OracleAS UDDI Registry
  - see *UDDI registry*
- oracle.soap.transport.allowUserInteraction property, 8-10, B-4
- oracle.ssl.ciphers property, 8-10, B-4
- oracle.wallet.password property, 8-10, B-4
- overview document URL
  - for UDDI, 10-20
- OverviewDoc data structure
  - UDDI Client API, 10-30

## P

---

- packageName request parameter, 8-4
- path configuration tag, 4-8
- pinging replication end-point, 10-44, 10-63
- PL/SQL stored procedures
  - setting up datasources, 5-8
  - writing, 5-2
- prefix configuration tag, 5-5
- proxy information
  - UDDI and, 10-29
- publisherAssertion
  - creating with UDDI Client API, 10-33
  - deleting relationship, 10-34
  - importing, 10-40
  - UDDI registry, 10-32
- publisherAssertion data structures, 10-2
- publishing API, 10-4, 10-28
- publishing end point
  - UDDI registry, 10-7
- publishing SOAP end point, 10-7
- publishing Web services, 10-4, 10-13
  - using Enterprise Manager, 10-13
- pushEnabled property, 10-44, 10-76
- pushTaskExecutionPeriod property, 10-44, 10-76

## Q

---

- queue-resource-ref configuration tag, 7-11
- quota groups
  - adding to UDDI registry, 10-38
  - associating publisher with, 10-39
  - deleting from UDDI registry, 10-39
  - UDDI registry and, 10-37
- quota limits
  - deleting, 10-39, 10-62
  - enforcing for UDDI registry, 10-37
  - retrieving, 10-39
  - setting, 10-38
  - viewing, 10-39

quotaLimitChecking property, 10-77

## R

---

receive-timeout configuration tag, 7-11

registry-based category validation, 10-44

adding category, 10-45

removing categories, 10-46

replication for UDDI, 10-41

enabling, 10-41

replication SOAP end point

for UDDI, 10-7

replicationEndPointSoapUrl, 10-64

reply-to-connection-factory-resource-ref configuration tag, 7-11

reply-to-queue-resource-ref configuration tag, 7-12

reply-to-topic-resource-ref configuration tag, 7-12

request header support

client-side, 12-3

server-side, 12-4

RPC typed request, 12-1

RPC untyped requests, 12-1

## S

---

saveBusiness method

UDDI Client API, 10-32

saveTModel method

UDDI Client API, 10-30

schema configuration tag, 5-5

schemaValidationUponIncomingRequests

property, 10-77

scope configuration tag, 3-9, 6-9

searches

UDDI registration, 10-1

UDDI registry, 10-9

using searching and browsing tool, 10-9

using UDDI Inquiry API, 10-10

security

HTTP transport properties

http.authRealm property, 8-9

http.authType property, 8-9

http.password property, 8-9

http.proxyAuthRealm property, 8-9

http.proxyAuthType property, 8-9

http.proxyHost property, 8-9

http.proxyPassword property, 8-9

http.proxyPort property, 8-9

http.proxyUsername property, 8-9

http.username property, 8-9

java.protocol.handler.pkgs property, 8-10

oracle.soap.transport.allowUserInteraction, 8-10

oracle.ssl.ciphers property, 8-10

oracle.wallet.location property, 8-10

oracle.wallet.password property, 8-10

server-side request header support, 12-4

session-timeout configuration tag, 3-9, 6-9

setAccessPoint method

UDDI Client API, 10-31

setBindingTemplates method

UDDI Client API, 10-31

setBusinessServices method

UDDI Client API, 10-31

setCategoryBag method

UDDI Client API, 10-30, 10-31

setContent method

UDDI Client API, 10-31

setFindQualifierStringList method

UDDI Client API, 10-12

setHttpProxy method, 10-29

setKeyName method

UDDI Client API, 10-30

setKeyValue method

UDDI Client API, 10-12, 10-30

setOperationalInfo option, 10-41, 10-66

setOverviewDoc method

UDDI Client API, 10-30

setProperty option, 10-36, 10-66

validation and, 10-45

setPublisherAssertions method

UDDI Client API, 10-34

setRoleQuotaLimits option, 10-39, 10-67

UDDI registry, 10-38

setTModelInstanceDetails method

UDDI Client API, 10-32

setTModelKey method

UDDI Client API, 10-12, 10-30

setUrlType method

UDDI Client API, 10-31

setWalletPassword option, 10-42, 10-67

Simple Object Access Protocol (SOAP), 10-3

SimpleAuthenticationLiaison method

UDDI Client API, 10-29

SOAP

features, 1-4, 1-5

web services, 1-4, 1-5

what is SOAP, 1-4, 1-5

SOAP end point

for UDDI inquiry, 10-7

for UDDI publishing, 10-7

SOAP header support, 12-3

SOAP request header support, 12-3, 12-4

SoapHTTPTransportLiaison interface

UDDI Client API, 10-29

soapRequestAuthMethod property, 10-44, 10-77

soapRequestTimeout property, 10-44, 10-78

SoapTransportLiaison

UDDI Client API, 10-10

SoapTransportLiaison instance

UDDI Client API, 10-29

startMaintainingUpdateJournal property, 10-42, 10-78

stateful document style, 6-3

stateful java class, 3-2

stateful-java-service configuration tag, 3-8, 6-8

stateless document style, 6-3

stateless java class, 3-2

stateless session EJBs

helloStatelessSession sample code, 4-1

- writing, 4-1
- stateless-java-service configuration tag, 3-8, 6-8
- stateless-session-ejb-service configuration tag, 4-7
- status property, 10-42, 10-43, 10-79
- stmtCacheSize property, 10-79
- stmtCacheType property, 10-79

## T

---

- taskExecutionPeriod property, 10-44, 10-80
- taxonomies
  - for UDDI, 10-1, 10-4, 10-5
- temporary-directory configuration tag, 3-8
- Thomas Register Supplier Identifier Code System, 10-5
- timer\_pool\_size property, 10-43, 10-80
- tModel data structures, 10-2
  - categorizing, 10-20
  - creating with
    - UDDI Client API, 10-30
    - UDDI publishing tool, 10-20
  - destroying, 10-40, 10-63
  - importing, 10-40
- tModel instance
  - creating, 10-31
- tModel operations
  - finding, 10-12
- tModelInstanceDetails instance
  - UDDI Client API, 10-31
- TModelInstanceInfo instance
  - UDDI Client API, 10-32
- tModelInstanceInfoKeyValidation property, 10-46, 10-80
- topic-resource-ref configuration tag, 7-12
- topic-subscription-name configuration tag, 7-12
- transferCustody option, 10-43, 10-68
- typed RPC requests, 12-1

## U

---

- UDDI Client API, 10-10, 10-28
- UDDI client library
  - using, 10-29
- UDDI publishing tool, 10-19
- UDDI registration, 10-3
  - searches, 10-1
- UDDI registry, 10-1, 10-3
  - adding categories, 10-15, 10-47
  - administration
    - administrative entity management, 10-39
    - built-in validated category management, 10-44, 10-46
    - command-line tool uddiadmin.jar, 10-34
    - database character set, 10-49
    - database configuration, 10-49
    - import operation, 10-40
    - performance monitoring and tuning, 10-48
    - server configuration, 10-35
    - transport security, 10-50
    - user account management, 10-36

- user groups, 10-36
- administration end point, 10-7
- classification support, 10-4
- classifications, 10-5
- configuration, 10-6, 10-7
- external validation, 10-47
  - adding category, 10-47
  - enabling, 10-47
  - removing category, 10-48
- identifiers, 10-5
- initialization, 10-6
- inquiry SOAP end point, 10-7
- listing configuration properties, 10-35
- production environment configuration, 10-8
- publishing SOAP end point, 10-7
- publishing Web Services
  - using Deploy Applications Wizard, 10-13
  - using publishing API, 10-28
- publishing Web services
  - using OracleAS UDDI publishing tool, 10-19
- registry administration
  - database character set, 10-49
- registry-based validation
  - adding tModel entity, 10-45
- removing category, 10-48
- replication, 10-4
- replication SOAP end point, 10-7
- setting operator name, 10-7
- setting properties, 10-7, 10-36
- standard taxonomies
  - ISO3166, 10-1
  - NAICS, 10-1
  - UNSPSC, 10-1
- structures, 10-2
- updating published Web services
  - using Web Services Details window, 10-16
- users, 10-36
- wallet endpoint, 10-7
- Web Services discovery, 10-4
  - using inquiry API, 10-10
  - using tools, 10-9
- Web Services publishing, 10-4
  - using Enterprise Manager, 10-13
  - using publishing API, 10-4, 10-28
  - using publishing tool, 10-19
- UDDI replication, 10-41
  - downloading configuration, 10-42, 10-63
  - enable update journal, 10-42
  - enabling, 10-41
  - enabling scheduling, 10-42
  - error handling, 10-43
  - number of threads, 10-43
  - scheduler, 10-43
  - scheduler status, 10-43
  - starting and stopping, 10-42
- uddiadmin.jar
  - registry administration command-line tool, 10-34
- UddiClient
  - closing instance, 10-32
  - creating query, 10-33

- initializing, 10-10
- initializing instance, 10-29, 10-32
- instance, 10-29
- UddiElementFactory instance
  - UDDI Client API, 10-11, 10-29, 10-30
- United Nations Standard Products and Services Codes (UNSPSC), 10-4
- Universal Discovery Description and Integration (UDDI)
  - see *UDDI*
- UNSPSC taxonomy, 10-4
- untyped RPC requests, 12-1
- update journal
  - for UDDI replication, 10-42
- uploadReplicationConfiguration option, 10-42, 10-68
- uri configuration tag, 3-9, 4-8, 5-5, 6-9, 7-12
- user management for UDDI registry, 10-36
- users
  - default for UDDI registry, 10-6
  - for UDDI registry, 10-36
- UUID
  - generating, 10-5

## V

---

- validate\_values SOAP Web service, 10-47
- validation
  - external, 10-47
    - adding category, 10-47
    - enabling, 10-47
    - removing category, 10-48

## W

---

- walletLocation property, 10-42, 10-81
- wallets
  - endpoint for UDDI, 10-7
  - location for UDDI, 10-42, 10-81
  - password for UDDI replication, 10-42
- Web Services
  - Bean support, 3-5, 4-4
  - client-side proxies, 8-2, 8-7
    - packageName request parameter, 8-4
  - discovering, 10-9
  - document style
    - deploying, 6-11
    - interface, 6-5
    - null value for Element, 6-2
    - stateful, 6-3
    - stateless, 6-3
  - encoding parameters, 3-11
  - encoding results, 3-11
  - generating client proxies, 8-5, A-6
  - generating WSDL documents, 9-3, A-7
  - home page, 8-2
  - Java Bean support, 3-5, 4-4
  - java class
    - deploying, 3-6, 3-11
    - interface, 3-4
    - preparing, 3-6

- stateful, 3-2
- stateless, 3-2
- supported parameter types, 3-5
- supported return value types, 3-5
- supported types, 3-3
- JMS
  - deploying, 7-13
  - preparing an EAR file, 7-13
- locating, 8-1
- PL/SQL stored procedures, 5-2
  - deploying, 5-9
  - preparing an EAR file, 5-8
  - setting up datasources, 5-8
- proxies, 8-7
- publishing, 10-13
- publishing using Enterprise Manager, 10-13
- serializing parameters, 3-11
- serializing results, 3-11
- stateless session EJBs, 4-1
  - bean code, 4-3
  - deploying, 4-6, 4-9
  - developing web services, 4-2
  - error handling, 4-4
  - home interface, 4-2, 4-3
  - preparing, 4-6
  - remote interface, 4-2
  - returning results, 4-4
  - sample code, 4-1
  - supported parameter types, 4-4
  - supported return value types, 4-4
- test page, 8-2
- updating published, 10-4
- WSDL descriptions, 8-2, 8-5
- Web Services Description Language (WSDL), 10-2
- Web Services registry
  - see *UDDI registry*
- WebServicesAssembler
  - DTD, 9-6
  - running, 9-1
  - sample input file, 9-1
  - sample output, 9-2
  - tag
    - class-name, 3-9, 6-8
    - connection-factory-resource-ref, 7-10
    - context, 3-7
    - db-pkg-name, 5-5
    - db-url, 5-5
    - description, 3-7
    - destination-path, 3-7
    - display-name, 3-7
    - ejb-name, 4-8
    - ejb-resource, 3-9, 4-8
    - interface-name, 3-9, 6-8
    - jar-generation, 5-5
    - java-resource, 3-9, 6-8
    - jms-delivery-mode, 7-10
    - jms-doc-service, 7-10
    - jms-expiration, 7-10
    - jms-message-type, 7-10
    - jms-priority, 7-11

- message-style, 3-9, 6-9
- method-name, 5-5
- operation, 7-11
- option name="force", 9-4
- option name="httpServerURL", 9-4
- option name="include-source", 8-6
- option name="packageIt", 9-4
- option name="source-path", 3-7
- option name="wsdl-location", 8-6
- option package-name, 8-6
- path, 4-8
- prefix, 5-5
- proxy-dir, 8-6
- queue-resource-ref, 7-11
- receive-timeout, 7-11
- reply-to-connection-factory-resource-ref, 7-11
- reply-to-queue-resource-ref, 7-12
- reply-to-topic-resource-ref, 7-12
- schema, 5-5
- scope, 3-9, 6-9
- session-timeout, 3-9, 6-9
- stateful-java-service, 3-8, 6-8
- stateless-java-service, 3-8, 6-8
- stateless-session-ejb-service, 4-7
- stateless-stored-procedure-java-service, 5-3
- temporary-directory, 3-8
- topic-resource-ref, 7-12
- topic-subscription-name, 7-12
- uri, 3-9, 4-8, 5-5, 6-9, 7-12
- wsdl-dir, 9-4
- WSDL file, 9-5
- writerXmlWriter instance
  - UDDI Client API, 10-32
- ws.debug property, 12-1
- WSDL file
  - getting directly, 8-3
- wsdl2java script, A-6

## X

---

- XmlWriter object
  - retrieving with UDDI publishing API, 10-32
  - UDDI Client API, 10-12