Oracle® Fusion Middleware

Introducing WebLogic Web Services for Oracle WebLogic Server

11*g* Release 1 (10.3.3)

E13759-02

April 2010

This document provides an introduction to WebLogic Web services for Oracle WebLogic Server, including interoperability and standards information.

Oracle Fusion Middleware Introducing WebLogic Web Services for Oracle WebLogic Server, 11*g* Release 1 (10.3.3)

E13759-02

Contents

4 Standards Supported by WebLogic Web Services

Preface

This preface describes the document accessibility features and conventions used in this guide—*Introducing WebLogic Web Services for Oracle WebLogic Server*.

Documentation Accessibility

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

Accessibility of Code Examples in Documentation

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

Accessibility of Links to External Web Sites in Documentation

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

Access to Oracle Support

Oracle customers have access to electronic support through My Oracle Support. For information, visit `http://www.oracle.com/support/contact.html` or visit `http://www.oracle.com/accessibility/support.html` if you are hearing impaired.

Conventions

The following text conventions are used in this document:

Convention	Meaning
boldface	Boldface type indicates graphical user interface elements associated with an action, or terms defined in text or the glossary.

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

1 Overview of WebLogic Web Services

The following sections provide an overview of WebLogic Web services as implemented by WebLogic Server:

1.1 What Are Web Services?

A Web service is a set of functions packaged into a single application that is available to other systems on a network. The network can be a corporate intranet or the Internet. Because Web services rely on basic, standard technologies which most systems provide, they are an excellent means for connecting distributed systems together. They can be shared by and used as a component of distributed Web-based applications. Other systems, such as customer relationship management systems, order-processing systems, and other existing back-end applications, can call a Web service function to request data or perform an operation.

Traditionally, software application architecture tended to fall into two categories: monolithic systems such as those that ran on mainframes or client-server applications running on desktops. Although these architectures worked well for the purpose the applications were built to address, they were closed and their functionality could not be incorporated easily into new applications.

As a result, the software industry has evolved toward loosely coupled service-oriented applications that interact dynamically over the Web. The applications break down the larger software system into smaller modular components, or shared services. These services can reside on different computers and can be implemented by vastly different technologies, but they are packaged and accessible using standard Web protocols, such as XML and HTTP, thus making them easily accessible by any user on the Web.

This concept of services is not new—RMI, COM, and CORBA are all service-oriented technologies. However, applications based on these technologies required them to use that particular technology, often from a particular vendor. This requirement typically hinders widespread integration of the application's functionality into other services on the network. To solve this problem, Web services are defined to share the following properties that make them easily accessible from heterogeneous environments:

- Web services are accessed using widely supported Web protocols such as HTTP.
- Web services describe themselves using an XML-based description language.
- Web services communicate with clients (both end-user applications or other Web services) through simple XML messages that can be produced or parsed by virtually any programming environment or even by a person, if necessary.

1.2 Why Use Web Services?

Major benefits of Web services include:

- Interoperability among distributed applications that span diverse hardware and software platforms
- Easy, widespread access to applications through firewalls using Web protocols
- A cross-platform, cross-language data model (XML) that facilitates developing heterogeneous distributed applications

Because you access Web services using standard Web protocols such as XML and HTTP, the diverse and heterogeneous applications on the Web (which typically already understand XML and HTTP) can automatically access Web services and communicate with each other.

These different systems can be Microsoft SOAP ToolKit clients, Java Platform, Enterprise Edition (Java EE) Version 5 applications, legacy applications, and so on. They are written in Java, C++, Perl, and other programming languages. Application interoperability is the goal of Web services and depends upon the service provider's adherence to published industry standards.

1.3 Anatomy of a WebLogic Web Service

WebLogic Web services are implemented according to the *JSR 109: Implementing Enterprise Web Services* specification (`http://www.jcp.org/en/jsr/detail?id=109`), which defines the standard Java EE runtime architecture for implementing Web services in Java. The specification also describes a standard Java EE Web service packaging format, deployment model, and runtime services, all of which are implemented by WebLogic Web services.

The following sections describe:

- Section 1.3.1, "The Programming Model—Metadata Annotations"
- Section 1.3.2, "The Development Model—Bottom-up and Top-down"

1.3.1 The Programming Model—Metadata Annotations

The *JSR 109: Implementing Enterprise Web Services* specification (`http://www.jcp.org/en/jsr/detail?id=109`) describes that a Java EE Web service is implemented by one of the following components:

- A Java class running in the Web container.
- A stateless session EJB running in the EJB container.

The code in the Java class or EJB implements the business logic of your Web service. Oracle recommends that, instead of coding the raw Java class or EJB directly, you use the JWS annotations programming model, which makes programming a WebLogic Web service much easier.

This programing model takes advantage of the new JDK 5.0 metadata annotations feature (described at `http://java.sun.com/developer/technicalArticles/releases/j2se15/`) in which you create an annotated Java file and then use Ant tasks to compile the file into a Java class and generate all the associated artifacts. The Java Web Service (JWS) annotated file is the core of your Web service. It contains the Java code that determines how your Web service behaves. A JWS file is an ordinary Java class file that uses annotations to specify the shape and characteristics of the Web service. The JWS annotations you can use in a JWS file include the standard ones defined by the *Web Services Metadata for the Java Platform* specification (`http://www.jcp.org/en/jsr/detail?id=181`) as well as a set of other standard or WebLogic-specific annotations, depending on the type of Web service you are creating.

This release of WebLogic Server supports both Java API for XML-Based Web services 2.1 (JAX-WS) Web services, described at `http://jcp.org/aboutJava/communityprocess/mrel/jsr224/index2.html`, and Java API for XML-Based RPC 1.1 (JAX-RPC) Web services, described at `https://jax-rpc.dev.java.net/`. JAX-RPC, an older specification, defined APIs and conventions for supporting XML Web services in the Java Platform as well support for the WS-I Basic Profile 1.0 to improve interoperability between JAX-RPC implementations. JAX-WS is a follow up to JAX-RPC 1.1. For more information, see Section 1.4, "How Do I Choose Between JAX-WS and JAX-RPC?."

Once you have coded the basic WebLogic Web service, you can program and configure additional advanced features. For example, you can specify that the SOAP messages be digitally signed and encrypted (as specified by the WS-Security specification at `http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=wss`). You configure these more advanced features of WebLogic Web services using WS-Policy files, which is an XML file that adheres to the WS-Policy specification and contains security-specific or Web service reliable messaging-specific XML elements that describe the security and reliable-messaging configuration, respectively. For information about the WS-Policy specification, see Section 4.18, "Web Services Policy Framework (WS-Policy) 1.5 and 1.2."

1.3.2 The Development Model—Bottom-up and Top-down

There are two approaches to Web service development: bottom-up and top-down. Each approach is described in the following sections.

1.3.2.1 Bottom-up Approach: Starting from Java

In the bottom-up approach, you develop your the JWS file from scratch. After you create the JWS file, you use the `jwsc` WebLogic Web service Ant task to compile the JWS file, as described by the *JSR 109: Implementing Enterprise Web Services* specification, described in Section 4.7, "JSR 109: Implementing Enterprise Web Services 1.2."

The `jwsc` Ant task always compiles the JWS file into a plain Java class; the only time it implements a stateless session EJB is if you implement a stateless session EJB in your JWS file. The `jwsc` Ant task also generates all the supporting artifacts for the Web service, packages everything into an archive file, and creates an Enterprise Application that you can then deploy to WebLogic Server.

By default, the `jwsc` Ant task packages the Web service in a standard Web application WAR file with all the standard WAR artifacts. The WAR file, however, contains additional artifacts to indicate that it is also a Web service; these additional artifacts include deployment descriptor files, the WSDL file that describes the public contract of the Web service, and so on. If you execute `jwsc` against more than one JWS file, you can choose whether `jwsc` packages the Web services in a single WAR file or each Web service in a separate WAR file. In either case, `jwsc` generates a single Enterprise Application.

If you implement a stateless session EJB in your JWS file, then the `jwsc` Ant task packages the Web service in a standard EJB JAR file with all the usual artifacts, such as the `ejb-jar.xml` and `weblogic-ejb.jar.xml` deployment descriptor files. The EJB JAR file also contains additional Web service-specific artifacts, as described in the preceding paragraph, to indicate that it is a Web service. Similarly, you can choose whether multiple JWS files are packaged in a single or multiple EJB JAR files.

For more information about the bottom-up approach, see the following sections:

- "Developing WebLogic Web Services Starting From Java: Main Steps" in *Getting Started With JAX-WS Web Services for Oracle WebLogic Server*.
- "Developing WebLogic Web Services Starting From Java: Main Steps" in *Getting Started With JAX-RPC Web Services for Oracle WebLogic Server*.

1.3.2.2 Top-down Approach: Starting from WSDL

In the top-down approach, you create the Web service from a WSDL file. You can use the `wsdlc` Ant task to generate a partial implementation of the Web service described by the WSDL file. The `wsdlc` Ant task generates he JWS service endpoint interface (SEI), the stubbed-out JWS class file, JavaBeans that represent the XML Schema data types, and so on, into output directories.

After running the `wsdlc` Ant task, (which typically you only do once) you update the generated JWS implementation file, for example, to add Java code to the methods so that they function as defined by your business requirements. The generated JWS implementation file does not initially contain any business logic because the `wsdlc` Ant task does not know how you want your Web service to function, although it does know the shape of the Web service, based on the WSDL file.

The `wsdlc` Ant task packages the JWS SEI and data binding artifacts together into a JAR file that you later specify to the `jwsc` Ant task. You never need to update this JAR file; the only file you update is the JWS implementation class.

For more information about the top-down approach, see the following sections:

- "Developing WebLogic Web Services Starting From a WSDL File: Main Steps" in *Getting Started With JAX-WS Web Services for Oracle WebLogic Server*.
- "Developing WebLogic Web Services Starting From a WSDL File: Main Steps" in *Getting Started With JAX-RPC Web Services for Oracle WebLogic Server*.

1.4 How Do I Choose Between JAX-WS and JAX-RPC?

As noted previously, this release of WebLogic Server supports the following Web services:

- Java API for XML-based Web Services 2.1 (JAX-WS), described in Section 4.5, "Java API for XML-based Web Services (JAX-WS) 2.1"
- Java API for XML-Based RPC 1.1 (JAX-RPC), described in Section 4.4, "Java API for XML-based RPC (JAX-RPC) 1.1"

Because JAX-WS is the successor to the JAX-RPC and it implements many of the new features in Java EE 5, Oracle recommends that you develop Web services with JAX-WS. JAX-RPC is considered legacy and the specification is no longer evolving.

The following table summarizes the benefits of choosing JAX-WS over JAX-RPC. There may be reasons to continue developing JAX-RPC Web services, which you can weigh against the benefits listed below. For additional documentation and examples about programming the features described in the following sections in a JAX-WS Web service, see the JAX-WS documentation available at `https://jax-ws.dev.java.net`. See also Chapter 4, "Standards Supported by WebLogic Web Services" for a comparison of the standards that are supported for JAX-WS and JAX-RPC.

Table 1–1 Benefits of JAX-WS

Benefit	Description
Data Binding Using JAXB 2.1	JAX-WS 2.1 fully supports the Java Architecture for XML Binding (JAXB) 2.1 specification (`http://jcp.org/en/jsr/detail?id=222`) and provides *full* XML Schema support. JAXB provides a convenient way to bind an XML schema to a representation in Java code. This makes it easy for you to incorporate XML data and processing functions in applications based on Java technology without having to know much about XML itself. For more information, see "Using JAXB Data Binding" in *Getting Started With JAX-WS Web Services for Oracle WebLogic Server*. By contrast, the built-in and user-defined data types you can use in a JAX-RPC-style Web service, although extensive, is limited to those described in "Understanding Data Binding" in *Getting Started With JAX-RPC Web Services for Oracle WebLogic Server*.
Document Attachments Using Streaming MTOM	Using MTOM and the `javax.activation.DataHandler` and `com.sun.xml.ws.developer.StreamingDataHandler` APIs you can specify that a Web service use a streaming API when reading inbound SOAP messages that include attachments, rather than the default behavior in which the service reads the entire message into memory. This feature increases the performance of Web services whose SOAP messages are particular large. For more information, see "Streaming SOAP Attachments" in *Programming Advanced Features of JAX-WS Web Services for Oracle WebLogic Server*.
Operate on messages at the XML level	The JAX-WS `javax.xml.ws.Provider` and `javax.xml.ws.Dispatch` APIs enable the Web service to operate at the XML message level, accessing the XML payloads directly
Web Service Annotations	The JAX-WS 2.1 programming model is very similar to JAX-RPC 1.1 Web services in that it uses metadata annotations described in the *JSR 181:Web Services Metadata for the Java Platform* specification (`http://jcp.org/en/jsr/detail?id=181`) and then Ant tasks to compile the annotated Java file into a deployable enterprise application (EAR) file. However, the JAX-WS 2.1 programming model is more robust because it defines additional annotations, listed in the JAX-WS 2.1 specification, that you can use to customize the mapping from Java to XML schema/WSDL and to map Web service operation parameter names to meaningful part/element names in the WSDL file. For a comparison of the Web service annotation support for JAX-WS and JAX-RPC, see "Web Service Annotation Support" in *WebLogic Web Services Reference*.

Table 1–1 (Cont.) Benefits of JAX-WS

Benefit	Description
XML-based Customizations	The JAX-WS 2.1 specification defines standard and portable XML-based customizations. These customizations, or binding declarations, can customize almost all WSDL components that can be mapped to Java, such as the service endpoint interface class, method name, parameter name, exception class, etc. Using binding declarations you can also control certain features, such as asynchrony, provider, wrapper style, and additional headers.
Logical and Protocol Handlers	JAX-WS 2.1 defines two types of handlers: logical and protocol handlers. While protocol handlers have access to an entire message such as a SOAP message, logical handlers deal only with the payload of a message and are independent of the protocol being used. Handler chains can now be configured on a per-port, per-protocol, or per-service basis. A new framework of context objects has been added to allow client code to share information easily with handlers.
EJB 3.0 Support	JAX-WS supports EJB 3.0. JAX-RPC supports EJB 2.1 only.

1.5 Roadmap for Implementing WebLogic Web Services

The following table provides a roadmap of common tasks for creating, deploying, and invoking WebLogic Web services.

Note: The JAX-WS implementation in Oracle WebLogic Server is extended from the JAX-WS Reference Implementation (RI) developed by the Glassfish Community (see `https://jax-ws.dev.java.net/`). All features defined in the JAX-WS specification (JSR-224) are fully supported by Oracle WebLogic Server.

The JAX-WS RI also contains a variety of extensions, provided by Glassfish contributors. Unless specifically documented, JAX-WS RI extensions are not supported for use in Oracle WebLogic Server.

Table 1–2 Roadmap for Implementing WebLogic Web Services

Major Task	Subtasks and Additional Information
Review supported standards	Chapter 4, "Standards Supported by WebLogic Web Services"
Run samples	▪ Section 2.1, "Samples for WebLogic Web Service Developers" ▪ JAX-WS Use Cases and Examples ▪ JAX-RPC Use Cases and Examples

Table 1–2 (Cont.) Roadmap for Implementing WebLogic Web Services

Major Task	Subtasks and Additional Information
Develop Web services using JAX-WS	■ *Getting Started With JAX-WS Web Services for Oracle WebLogic Server* ■ Use Cases and Examples ■ Invoking a Web Service Using Asynchronous Request-Response ■ Publishing a Web Service Endpoint ■ Using Callbacks ■ Optimizing Binary Data Transmission Using MTOM/XOP ■ Creating Dynamic Proxy Classes ■ Using XML Catalogs ■ Creating and Using SOAP Message Handlers ■ Programming RESTful Web Services ■ Publishing and Finding Web Services Using UDDI
Develop Web services using JAX-RPC	■ *Getting Started With JAX-RPC Web Services for Oracle WebLogic Server* ■ Use Cases and Examples ■ Invoking a Web Service Using Asynchronous Request-Response ■ Using Web Services Reliable Messaging ■ Creating Conversational Web Services ■ Using the Asynchronous Features Together ■ Using Callbacks to Notify Clients of Events ■ Sending Binary Data Using MTOM/XOP ■ Creating Buffered Web Services ■ Using JMS Transport as the Connection Protocol ■ Creating and Using SOAP Message Handlers ■ Using Database Web Services ■ Publishing and Finding Web Services Using UDDI
Secure the Web Service	■ Configuring Message-Level Security ■ Configuring Transport-Level Security ■ Configuring Access Control Security (JAX-RPC Only)
Upgrade	■ **JAX-WS**: No steps are required to upgrade JAX-WS to 10.3.1. ■ **JAX-RPC**: Upgrading WebLogic Web services From Previous Releases to 10.3.1

1.6 Using Oracle IDEs to Build Web Services

The following Oracle IDE tools are available to build Web services:

- **Oracle JDeveloper**—Oracle's full-featured Java IDE, can be used for end-to-end development of Web services. Developers can build Java classes or EJBs, expose them as Web services, automatically deploy them to an instance of Oracle WebLogic Server, and immediately test the running Web service. Alternatively, JDeveloper can be used to drive the creation of Web services from WSDL descriptions. JDeveloper also is Ant-aware. You can use this tool to build and run Ant scripts for assembling the client and for assembling and deploying the service. For more information, see the Oracle JDeveloper online help. For information about installing JDeveloper, see *Oracle Fusion Middleware Installation Guide for Oracle JDeveloper*.

- **Oracle Enterprise Pack for Eclipse (OEPE)**—Provides a collection of plug-ins to the Eclipse IDE platform that facilitate development of WebLogic Web services. For more information, see the Eclipse IDE platform online help.

1.7 New and Changed Features in this Release

For a comprehensive listing of the new WebLogic Server Web service features introduced in this release, see "Web Services" in *What's New in Oracle WebLogic Server*.

2

Samples and Related Information

The following sections describe the samples and related information that is available to assist you in learning more about WebLogic Web services.

- Section 2.1, "Samples for WebLogic Web Service Developers"
- Section 2.2, "WebLogic Web Services Documentation Set"
- Section 2.3, "Related Documentation—WebLogic Server Application Development"

2.1 Samples for WebLogic Web Service Developers

In addition to this document, Oracle provides a variety of code samples for Web services developers. The samples and tutorials illustrate WebLogic Web services in action, and provide practical instructions on how to perform key Web service development tasks.

Oracle recommends that you run the Web service samples before programming your own application that use Web services.

2.1.1 Web Services Samples in the WebLogic Server Distribution

WebLogic Server optionally installs API code samples in `WL_HOME\samples\server\examples\src\examples\webservices`, where *WL_HOME* is the top-level directory of your WebLogic Server installation. You can start the samples server, and obtain information about the samples and how to run them from the WebLogic Server Start menu.

2.1.2 Avitek Medical Records Application (MedRec) and Tutorials

MedRec is an end-to-end sample Java EE application shipped with WebLogic Server that simulates an independent, centralized medical record management system. The MedRec application provides a framework for patients, doctors, and administrators to manage patient data using a variety of different clients.

MedRec demonstrates WebLogic Server and Java EE features, and highlights Oracle-recommended best practices. MedRec is included in the WebLogic Server distribution, and can be accessed from the Start menu on Windows machines. For Linux and other platforms, you can start MedRec from the `WL_HOME\samples\domains\medrec` directory, where *WL_HOME* is the top-level installation directory for WebLogic Server. A Spring version of the application is accessible from the `WL_HOME\samples\domains\medrec-spring` directory.

2.1.3 Additional Web Services Samples Available for Download

Additional API samples for download can be found at `https://codesamples.samplecode.oracle.com/`. These samples include Oracle-certified ones, as well as samples submitted by fellow developers.

2.2 WebLogic Web Services Documentation Set

This document is part of a larger WebLogic Web services documentation set that covers a comprehensive list of Web services topics. The full documentation set includes the documents summarized in the following table.

Table 2–1 WebLogic Web Services Documentation Set

This document . . .	Describes . . .
Introducing WebLogic Web Services for Oracle WebLogic Server (This Document)	An introduction to WebLogic Web services, the standards that are supported, interoperability information, and relevant samples and documentation.
Getting Started With JAX-WS Web Services for Oracle WebLogic Server	The basic knowledge and tasks required to program a simple WebLogic Web service using JAX-WS. The guide includes use cases and examples, iterative development procedures, typical JWS programming steps, data type information, and how to invoke a Web service.
Programming Advanced Features of JAX-WS Web Services for Oracle WebLogic Server	How to program more advanced features using JAX-WS, such as callbacks, XML Catalog, and SOAP message handlers.
Getting Started With JAX-RPC Web Services for Oracle WebLogic Server	The basic knowledge and tasks required to program a simple WebLogic Web service using JAX-RPC. The guide includes use cases and examples, iterative development procedures, typical JWS programming steps, data type information, and how to invoke a Web service.
Programming Advanced Features of JAX-RPC Web Services for Oracle WebLogic Server	How to program more advanced features using JAX-RPC, such as Web service reliable messaging, callbacks, conversational Web services, use of JMS transport to invoke a Web service, and SOAP message handlers.
Securing WebLogic Web Services for Oracle WebLogic Server	How to program and configure message-level (digital signatures and encryption), transport-level, and access control security for a Web service.
WebLogic Web Services Reference for Oracle WebLogic Server	Reference information on JWS annotations, Ant tasks, reliable messaging WS-Policy assertions, security WS-Policy assertions, and deployment descriptors.

2.3 Related Documentation—WebLogic Server Application Development

For comprehensive guidelines for developing, deploying, and monitoring WebLogic Server applications, refer to the documents summarized in the following table.

Table 2–2 Related Documentation—WebLogic Server Application Development

Review this document . . .	To learn how to . . .
Introducing Web Services	Develop Web services for Oracle Fusion Middleware 11*g*.
Security and Administrator's Guide for Web Services	Administer Web services for Oracle Fusion Middleware 11*g*.
Developing Applications for Oracle WebLogic Server	Develop WebLogic Server components (such as Web applications and EJBs) and applications.

Table 2–2 (Cont.) Related Documentation—WebLogic Server Application Development

Review this document . . .	To learn how to . . .
Developing Web Applications, Servlets, and JSPs for Oracle WebLogic Server	Develop Web applications, including servlets and JSPs, that are deployed and run on WebLogic Server.
Programming WebLogic Enterprise JavaBeans for Oracle WebLogic Server	Develop EJBs that are deployed and run on WebLogic Server.
Programming XML for Oracle WebLogic Server	Design and develop applications that include XML processing.
Deploying Applications to Oracle WebLogic Server	Deploy WebLogic Server applications. Use this guide for both development and production deployment of your applications.
"Configuring Applications for Production Deployment" in *Deploying Applications to Oracle WebLogic Server*	Configure your applications for deployment to a production WebLogic Server environment.
Performance and Tuning for Oracle WebLogic Server	Monitor and improve the performance of WebLogic Server applications.
"Overview of WebLogic Server System Administration" in *Introduction to Oracle WebLogic Server*	Administer WebLogic Server and its deployed applications.

3

Interoperability with Microsoft WCF/.NET

In conjunction with Microsoft, Oracle has performed interoperability testing to ensure that the Web services created using WebLogic Server can access and consume Web services created using Microsoft Windows Communication Foundation (WCF)/.NET 3.0 and 3.5 Framework and vice versa. For more information, see `http://msdn2.microsoft.com/en-us/netframework/default.aspx`.

Interoperability tests were completed on JAX-WS and JAX-RPC Web services in the following areas:

Table 3–1 Completed Interoperability Tests

Area	Interoperability Guidelines
Basic and complex data types	Section 3.1, "Basic Data Types Interoperability Guidelines"
WS-I Basic Profile 1.1	Section 3.2, "Basic Profile 1.1 Interoperability Guidelines"
Web Services Security (WS-Security) 1.0 and 1.1	Section 3.3, "WS-Security Interoperability Guidelines"
Web Services Security Policy (WS-SecurityPolicy) 1.2	Section 3.4, "WS-SecurityPolicy Interoperability Guidelines"
Web Services Secure Conversation Language (WS-SecureConversation) 1.3	Section 3.5, "WS-SecureConversation Interoperability Guidelines"
Web Services Policy Framework (WS-Policy) 1.5	No interoperability guidelines provided.
Web Services Addressing (WS-Addressing) 0.9 and 1.0	N/A
Message Transmission Optimization Mechanism (MTOM)	N/A
Web Services Reliable Messaging (WS-ReliableMessaging) 1.0 and 1.1	Section 3.6, "WS-ReliableMessaging Interoperability Guidelines" **Note**: Tested for JAX-RPC only.
Web Services Trust (WS-Trust) 1.3	Section 3.7, "WS-Trust Interoperability Guidelines"

In addition, the following combined features were tested:

- MTOM and WS-Security
- WS-ReliableMessaging and MTOM (JAX-RPC only)
- WS-ReliableMessaging 1.1 and WS-Addressing 0.9 and 1.0 (JAX-RPC only)

- WS-ReliableMessaging 1.0 and WS-Addressing 0.9 and 1.0 (JAX-RPC only)
- WS-ReliableMessaging 1.1 and WS-SecureConversation 1.3 (JAX-RPC only)
- WS-ReliableMessaging 1.0 and WS-SecureConversation 1.3 (JAX-RPC only)
- WS-Policy 1.5 and WS-SecurityPolicy 1.2

The following sections describe the interoperability issues and guidelines that were identified during the testing.

3.1 Basic Data Types Interoperability Guidelines

When using the `anyType` class with Microsoft .NET 3.0/3.5 the Java data type returned cannot be guaranteed. If a specific Java data type is required, avoid using `anyType`.

3.2 Basic Profile 1.1 Interoperability Guidelines

JAX-WS 2.0 enforces strict WS-I Basic Profile 1.1 compliance. Microsoft .NET 3.0/3.5 framework does not enforce string Basic Profile 1.1 semantics for the use case described on the Sun Java Web site at:
`http://java.sun.com/webservices/reference/tutorials/wsit/doc/DataBinding7.html`

3.3 WS-Security Interoperability Guidelines

The following lists interoperability guidelines for WS-Security:

- Use of `<sp:Strict>` layout assertions (shown below) cannot be guaranteed.

```
<sp:Layout>
   <wsp:Policy>
      <sp:Strict/>
   </wsp:Policy>
</sp:Layout>
```

  Instead, you should define your policy as follows:

```
<sp:Layout>
   <wsp:Policy>
      <sp:Lax/>
   </wsp:Policy>
</sp:Layout>
```

- The following assertions are not supported by Microsoft .NET 3.0/3.5:
  - Digest password in `UsernameToken`
  - `<sp:EncryptedSupportingTokens>`
  - Element-level signature
  - Element-level encryption
- Support of asymmetric binding for WS-Security 1.1 cannot be guaranteed on Microsoft .NET 3.0/3.5.

3.4 WS-SecurityPolicy Interoperability Guidelines

In this release, WebLogic Server and Microsoft .NET 3.5 support Web Services Security Policy (WS-SecurityPolicy) 1.2. Microsoft .NET 3.0 supports the December 2005 draft version of the WS-SecurityPolicy specification.

In the December 2005 draft version of the specification, the `<sp:SignedEncryptedSupportingTokens>` policy assertion is not supported. As a result, Microsoft .NET 3.0 encrypts the `UsernameToken` in the `<sp:SignedSupportingTokens>` policy assertion. If you use the `<sp:SignedSupportingTokens>` policy assertion without encrypting the `UsernameToken`, the WebLogic Server and Microsoft .NET Web services will not interoperate.

3.5 WS-SecureConversation Interoperability Guidelines

The following lists interoperability guidelines for WS-SecureConversation:

- Use of WS-SecureConversation token for HTTPS authentication (in `<sp:EndorsingSupportingTokens>`) is not supported.
- Oracle recommends that you do not use `<sp:EncryptBeforeSigning/>` unless there is a security requirement. Instead, use `<sp:SignBeforeEncrypt>` (the default).
- Although WebLogic Server Web services support cookie mode conversations, this feature is a Microsoft proprietary implementation, and may not be supported by other vendors.
- When using `<sp:BootstrapPolicy>` policy assertion, you should refer to the guidelines defined in Section 3.3, "WS-Security Interoperability Guidelines."
- There is no standard method of supporting cancel and renew of WS-SecureConversation defined in the WS-SecurityPolicy or WS-SecureConversation specifications. The method used by Microsoft .NET to support cancel and renew of WS-SecureConversation is not compatible with WebLogic Server 10.*x*. As a result:
  - For a Microsoft .NET client to interoperate with a WebLogic Server Web service, the `Compatibility` flag must be set on the server side via the Web service Security MBean using the `setCompatibilityPreference("msft")` method.
  - For a WebLogic Server Web service client to interoperate with a WebLogic Server Web service that has the `Compatibility` flag set, the client must set this flag as well, as follows:

    ```
    stub._setProperty(WLStub.POLICY_COMPATIBILITY_PREFERENCE,"msft");
    ```

3.6 WS-ReliableMessaging Interoperability Guidelines

The following lists interoperability guidelines for WS-ReliableMessaging:

- Anonymous request/response is not defined in the WS-ReliableMessaging specification (`http://docs.oasis-open.org/ws-rx/wsrm/200702/wsrm-1.1-spec-os-01.pdf`) and is, consequently, not supported by WebLogic Server.

- For WS-ReliableMessaging security, you must use WS-SecureConversation as per the guidelines in the *WS-I Reliable Secure Profile Version 1.0 Working Group Draft* specification at `http://www.ws-i.org/Profiles/ReliableSecureProfile-1.0.html`.
- Asynchronous reliable messaging plus WS-SecureConversation or WS-Trust is not supported for WebLogic Web service clients and Microsoft .NET services.

3.7 WS-Trust Interoperability Guidelines

WebLogic Server does not interoperate with Microsoft .NET according to the Microsoft .NET interoperability scenarios that use both SAML and WS-Trust, as defined in the following documents on the Microsoft .NET Web Services Interoperability Plug-Fest Web site at `http://mssoapinterop.org/ilab/`:

- WS-SX Scenarios Document at `http://mssoapinterop.org/ilab/Trust13/WCFInteropPlugFest_WSTrust13.doc`
- WCF (Indigo) Interoperability Lab: WS-Trust10 Scenarios at `http://mssoapinterop.org/ilab/Trust10/WCFInteropPlugFest_WSTrust10.doc`

With the proper configuration, however, WebLogic Server can interoperate with Microsoft .NET. For example, with proper configuration on STS and Microsoft .NET client, it can interoperate with WebLogic server with SAML 1.1 Token Profile 1.0 in accordance with WS-Security 1.0. More details are provided in the following sections:

- Section 3.7.1, "Configuring Microsoft .NET STS for WS-Trust"
- Section 3.7.2, "Configuring WebLogic Web Service"
- Section 3.7.3, "Configuring the Microsoft .NET Client"
- Section 3.7.4, "Using SAML Assertions Referenced from SignedInfo"

3.7.1 Configuring Microsoft .NET STS for WS-Trust

Microsoft .NET requires a Security Token Service (STS) to generate a SAML Assertion and supports WS-Security SAML 1.1 Token Profile for WS-Security 1.0. Microsoft .NET favors symmetric bindings for WS-Security and SAML holder-of-key confirmation methods. Oracle WebLogic, on the other hand, performs better using asymmetric bindings and SAML sender-vouches confirmation methods, using X.509 certificates to sign the message and bind the SAML assertion.

On the WebLogic Server side, the inbound policy is WS-Security SAML 1.1 Token Profile for WS-Security 1.0 and the outbound policy ensures that the message is signed by the Web service. This is required because Microsoft .NET expects that an endpoint that is protected using WS-Security will secure the response as well. To support this configuration, the WebLogic Server Security realm needs to be configured to consume and validate SAML Assertions as well as configure Public/Private Key pairs and corresponding trust stores for the message signature operations dictated by the WS-Security policies.

The flow is as follows:

1. The Microsoft .NET client calls the STS.
2. The STS generates a SAML Assertion signed by the STS that contains the name of the user as the Subject.

The SAML Assertion uses the sender-vouches confirmation method.

3. The SAML Assertion is added to the WS-Security Header, and the message is signed by the invoking service.
4. The message is sent to WebLogic Server where the SAML Assertion is verified along with the message signature.
5. Once the message is processed, the return message is signed by the WebLogic server's identity.
6. The signature is validated by the Microsoft .NET client to ensure that the message has not been tampered and was sent by the WebLogic server.

STS needs to be configured, as follows:

- Use X509RawCertificate format for WSS1.0 instead of SHA1 Thumbprint.
- Use Sender-Vouches confirmation method instead of Holder of Key.
- Sign the assertion with the private key of the issuer, not the encrypted key. Use of the unencrypted asymmetric keys over symmetric encrypted keys improves performance.
- Include an AuthenticationStatement in the SAML Assertion. WebLogic uses this statement to access the user's identity.
- Add a wsu:Id to the saml:Assertion. Otherwise the Microsoft .NET client cannot use it as an IssuedToken with an asymmetric binding.

3.7.2 Configuring WebLogic Web Service

For the WebLogic Web service, use the predefined Wssp1.2-2007-Saml1.1-SenderVouches-Wss1.0.xml to enforce SAML 1.1 sender-vouches authentication with WS-Security 1.0 for message binding.

3.7.3 Configuring the Microsoft .NET Client

Configure the Microsoft .NET client as shown below. The SAML token is retrieved from the STS, so the Microsoft .NET client needs to be configured to communicate with STS. Authentication using a token retrieved from an STS is called an IssuedToken.

```
?xml version="1.0"?>
<wsp:Policy
  xmlns:wsp="http://schemas.xmlsoap.org/ws/2004/09/policy"
  xmlns:sp="http://docs.oasis-open.org/ws-sx/ws-securitypolicy/200702"
  >
  <sp:AsymmetricBinding>
    <wsp:Policy>
      <sp:InitiatorToken>
        <wsp:Policy>
          <sp:X509Token
            sp:IncludeToken=
"http://docs.oasis-open.org/ws-sx/ws-securitypolicy/200702/IncludeToken/AlwaysToRe
cipient">
            <wsp:Policy>
              <sp:WssX509V3Token10/>
            </wsp:Policy>
          </sp:X509Token>
        </wsp:Policy>
      </sp:InitiatorToken>
      <sp:RecipientToken>
        <wsp:Policy>
```

```
          <sp:X509Token
            sp:IncludeToken=
"http://docs.oasis-open.org/ws-sx/ws-securitypolicy/200702/IncludeToken/Never">
            <wsp:Policy>
              <sp:WssX509V3Token10/>
            </wsp:Policy>
          </sp:X509Token>
        </wsp:Policy>
      </sp:RecipientToken>
      <sp:AlgorithmSuite>
        <wsp:Policy>
          <sp:Basic256/>
        </wsp:Policy>
      </sp:AlgorithmSuite>
      <sp:Layout>
        <wsp:Policy>
          <sp:Lax/>
        </wsp:Policy>
      </sp:Layout>
      <sp:IncludeTimestamp/>
      <sp:ProtectTokens/>
      <sp:OnlySignEntireHeadersAndBody/>
    </wsp:Policy>
  </sp:AsymmetricBinding>
  <sp:SignedSupportingTokens>
    <wsp:Policy>
      <sp:SamlToken
        sp:IncludeToken=
"http://docs.oasis-open.org/ws-sx/ws-securitypolicy/200702/IncludeToken/AlwaysToRe
cipient">
        <wsp:Policy>
          <sp:WssSamlV11Token10/>
        </wsp:Policy>
      </sp:SamlToken>
    </wsp:Policy>
  </sp:SignedSupportingTokens>
  <sp:Wss10>
    <wsp:Policy>
      <sp:MustSupportRefKeyIdentifier/>
      <sp:MustSupportRefIssuerSerial/>
    </wsp:Policy>
  </sp:Wss10>
</wsp:Policy>
```

3.7.4 Using SAML Assertions Referenced from SignedInfo

When the SAML assertion is referenced in the `<ds:SignedInfo>` element of a `<ds:Signature>` element in a `<wsee:Security>` header, Microsoft .NET does not support a SAML assertion that is referenced from `<wsse:SecurityTokenReference>`. Use of <wsse:SecurityTokenReference> is defined as a best practice in the WS-Security specification, described at `http://www.oasis-open.org/committees/download.php/16768/wss-v1.1-spec-os-SAMLTokenProfile.pdf`.

For compatibility with Microsoft .NET, you must set the `WLStub.POLICY_COMPATIBILITY_PREFERENCE` flag to `WLStub.POLICY_COMPATIBILITY_MSFT` flag in Web service client code. When the flag is set, the SAML assertion will be signed with direct reference, rather than using a `SecurityTokenReference`.

The following provides an example of how to set the Microsoft .NET compatibility flag for a JAX-WS Web service client:

Example 3–1 Setting the Microsoft .NET Compatibility Flag in a JAX-WS Web Service Client

```
. . .
import weblogic.wsee.jaxrpc.WLStub;
. . .
public String test(String hello) throws Exception {
     . . .
    BindingProvider provider = (BindingProvider)port;
    Map context = provider.getRequestContext();
     . . .
     . . .
    context.put(WLStub.POLICY_COMPATIBILITY_PREFERENCE, WLStub.POLICY_COMPATIBILITY_MSFT);
    try {
       String result = port.getName(hello);
       System.out.println("MSFT Result was: " + result);
       return result;
      } catch (Exception e) {
        throw new RuntimeException (e);
      }
}
```

The following provides an example of how to set the Microsoft .NET compatibility flag for a JAX-RPC Web service client:

Example 3–2 Setting the Microsoft .NET Compatibility Flag in a JAX-RPC Web Service Client

```
. . .
. . .
import weblogic.wsee.jaxrpc.WLStub;
. . .

@WebMethod()
public String callSamlHelloSV_WSS10_MSFT(String input) {
     try {
         System.out.println("Calling sayHello(" + input + ") with MSFT Ways");
         ((Stub)port)._setProperty(WLStub.POLICY_COMPATIBILITY_PREFERENCE,
                       WLStub.POLICY_COMPATIBILITY_MSFT);
         String result = port.sayHelloSV_WSS10(input);
         System.out.println("MSFT Result was: " + result);
         return result;
     }
     catch (RemoteException e) {
         throw new RuntimeException(e);
     }
}
```

4

Standards Supported by WebLogic Web Services

Many specifications that define Web service standards are written so as to allow for broad use of the specification throughout the industry. The Oracle implementation of a particular specification might not cover all possible usage scenarios covered by the specification.

Oracle considers interoperability of Web service platforms to be more important than providing support for all possible edge cases of the Web service specifications. Oracle complies with the following specifications from the Web Services Interoperability Organization and considers them to be the baseline for Web services interoperability:

- *Basic Profile 1.1*:
  `http://www.ws-i.org/Profiles/BasicProfile-1.1-2004-08-24.html`
- *Basic Security Profile 1.1*:
  `http://www.ws-i.org/Profiles/BasicSecurityProfile-1.1.html`

This guide does not necessarily document all of the *Basic Profile 1.1* and *Basic Security Profile 1.1* requirements. This guide does, however, document features that are beyond the requirements of the *Basic Profile 1.1* and *Basic Security Profile 1.1*.

The following table summarizes the Web service specifications that are part of the Oracle implementation, organized by high-level feature.

Table 4–1 Oracle Implementation of Web Service Specifications

Feature	Specification	JAX-WS	JAX-RPC
Programming model (based on metadata annotations) and runtime architecture	**JSR 109: Implementing Enterprise Web Services 1.2**—Programming model and runtime architecture for implementing Web services in Java that run on a Java EE application server, such as WebLogic Server. See Section 4.7, "JSR 109: Implementing Enterprise Web Services 1.2."	Version 1.2	Version 1.2
Programming model (based on metadata annotations) and runtime architecture	**Web Services Metadata for the Java Platform 2.0 (JSR-181)**—Standard annotations that you can use in your Java Web service (JWS) file to facilitate the programming of Web services. See Section 4.16, "Web Services Metadata for the Java Platform 2.0 (JSR-181)."	Supports	Supports
Programming APIs	**Java API for XML-based Web Services (JAX-WS) 2.1**—Standards-based API for coding, assembling, and deploying Java Web services. The integrated stack includes JAX-WS 2.1, JAXB 2.1, and SAAJ 1.3. See Section 4.5, "Java API for XML-based Web Services (JAX-WS) 2.1."	Version 2.1	N/A

Table 4–1 (Cont.) Oracle Implementation of Web Service Specifications

Feature	Specification	JAX-WS	JAX-RPC
Programming APIs	**Java API for XML-based RPC (JAX-RPC) 1.1**—Java APIs for making XML-based remote procedure calls (RPC). See Section 4.4, "Java API for XML-based RPC (JAX-RPC) 1.1."	N/A	Version 1.1
Data binding	**Java Architecture for XML Binding (JAXB) 2.1**—Implementation used to bind an XML schema to a representation in Java code. JAXB is supported by JAX-WS Web services only. See Section 4.6, "Java Architecture for XML Binding (JAXB) 2.1."	Version 2.1	N/A
Data binding	**Apache XMLBeans 2.0**—A technology for binding XML schema to Java types and for accessing XML data in a variety of ways. XMLBeans is the default binding technology for JAX-RPC Web services. See Section 4.2, "Apache XMLBeans 2.0."	N/A	2.0
Web service description	**Web Services Description Language (WSDL) 1.1**—XML-based specification that describes a Web service. See Section 4.15, "Web Services Description Language (WSDL) 1.1."	Version 1.1	Version 1.1
Web service description	**Web Services Policy Framework (WS-Policy) 1.5 and 1.2**—General purpose model and corresponding syntax to describe and communicate the policies of a Web service. See Section 4.18, "Web Services Policy Framework (WS-Policy) 1.5 and 1.2."	Versions 1.5 and 1.2	Versions 1.5 and 1.2
Web service description	**Web Services Policy Attachment (WS-PolicyAttachment) 1.5 and 1.2**—Abstract model and an XML-based expression grammar for policies. See Section 4.17, "Web Services Policy Attachment (WS-Policy Attachment) 1.5 and 1.2."	Versions 1.5 and 1.2	Versions 1.5 and 1.2
Data exchange between Web service and requesting client	**Simple Object Access Protocol (SOAP) 1.1 and 1.2**—Lightweight XML-based protocol used to exchange information in a decentralized, distributed environment. See Section 4.10, "Simple Object Access Protocol (SOAP) 1.1 and 1.2."	Versions 1.1 and 1.2	Versions 1.1 and 1.2
Data exchange between Web service and requesting client	**SOAP with Attachments API for Java (SAAJ) 1.3**—Implementation that developers can use to produce and consume messages conforming to the SOAP 1.1 specification and SOAP with Attachments notes. See Section 4.11, "SOAP with Attachments API for Java (SAAJ) 1.3."	Version 1.3	Version 1.3
Security	**Web Services Security (WS-Security) 1.1 and 1.0**—Standard set of SOAP [SOAP11, SOAP12] extensions that can be used when building secure Web services to implement message content integrity and confidentiality. See Section 4.22, "Web Services Security (WS-Security) 1.1 and 1.0."	Versions 1.1 and 1.0	Versions 1.1 and 1.0
Security	**Web Services Security Policy (WS-SecurityPolicy) 1.2**—Set of security policy assertions for use with the WS-Policy framework. See Section 4.23, "Web Services Security Policy (WS-SecurityPolicy) 1.2."	Version 1.2	Version 1.2
Security	**Security Assertion Markup Language (SAML) 2.0 and 1.1**—XML standard for exchanging authentication and authorization data between security domains. See Section 4.8, "Security Assertion Markup Language (SAML) 2.0 and 1.1."	Versions 2.0 and 1.1	Versions 2.0 and 1.1

Table 4–1 (Cont.) Oracle Implementation of Web Service Specifications

Feature	Specification	JAX-WS	JAX-RPC
Security	**Security Assertion Markup Language (SAML) Token Profile 1.1 and 1.0**—Set of SOAP extensions that implement SOAP message authentication and encryption. See Section 4.9, "Security Assertion Markup Language (SAML) Token Profile 1.1 and 1.0."	Versions 1.1 and 1.0	Versions 1.1 and 1.0
Reliable communication	**Web Services Addressing (WS-Addressing) 1.0**—Transport-neutral mechanisms to address Web services and messages. See Section 4.12, "Web Services Addressing (WS-Addressing) 1.0."	Version 1.0	Version 1.0
Reliable communication	**Web Services Reliable Messaging (WS-ReliableMessaging) 1.1**—Implementation that enables two Web services running on different WebLogic Server instances to communicate reliably in the presence of failures in software components, systems, or networks. This specification is supported for JAX-RPC only. See Section 4.19, "Web Services Reliable Messaging (WS-ReliableMessaging) 1.1."	N/A	Version 1.1
Reliable communication	**Web Services Reliable Messaging Policy Assertion (WS-RM Policy) 1.1**—Domain-specific policy assertion for reliable messaging for use with WS-Policy and WS-ReliableMessaging. See Section 4.20, "Web Services Reliable Messaging Policy Assertion (WS-RM Policy) 1.1."	N/A	Version 1.1
Reliable communication	**Web Services Trust Language (WS-Trust) 1.3**—Extensions that build on *Web Services Security (WS-Security) 1.1* to secure asynchronous communication. See Section 4.24, "Web Services Trust Language (WS-Trust) 1.3."	Version 1.3	Version 1.3
Reliable communication	**Web Services Secure Conversation Language (WS-SecureConversation) 1.3**—Extensions that build on *Web Services Security (WS-Security) 1.1* and *Web Services Trust Language (WS-Trust) 1.3* to secure asynchronous communication. See Section 4.21, "Web Services Secure Conversation Language (WS-SecureConversation) 1.3."	Version 1.3	Version 1.3
Atomic transactions	**Web Services Atomic Transaction**—Defines the Atomic Transaction coordination type that is to be used with the extensible coordination framework described in the Web Services Coordination specification. The WS-AtomicTransaction and WS-Coordination specifications define an extensible framework for coordinating distributed activities among a set of participants. See Section 4.13, "Web Services Atomic Transaction (WS-AtomicTransaction) Version 1.2, 1.1, and 1.0".	Versions 1.2, 1.1, and 1.0	N/A

Table 4–1 (Cont.) Oracle Implementation of Web Service Specifications

Feature	Specification	JAX-WS	JAX-RPC
Atomic transactions	**Web Services Coordination**—Defines an extensible framework for providing protocols that coordinate the actions of distributed applications. The WS-AtomicTransaction and WS-Coordination specifications define an extensible framework for coordinating distributed activities among a set of participants. See Section 4.14, "Web Services Coordination (WS-Coordination) Version 1.2, 1.1, and 1.0".	Versions 1.2, 1.1, and 1.0	N/A
Advertisement (registration and discovery)	**Universal Description, Discovery, and Integration (UDDI) 2.0**—Standard for describing a Web service; registering a Web service in a well-known registry; and discovering other registered Web services. See Section 4.25, "Universal Description, Discovery, and Integration (UDDI) 2.0."	Version 2.0	Version 2.0
Advertisement (registration and discovery)	**Java API for XML Registries (JAXR) 1.0**—Uniform and standard Java API for accessing different kinds of XML Registries. See Section 4.3, "Java API for XML Registries (JAXR) 1.0."	Version 1.0	Version 1.0

The following sections describe the specifications in more detail. Specifications are listed in alphabetical order. Additional specifications that WebLogic Web services support are listed in Section 4.26, "Additional Specifications Supported by WebLogic Web Services."

4.1 A Note About JAX-WS 2.1 RI/JDK 6.0 Extensions

A subset of the APIs described in this document (such as `com.sun.xml.ws.developer` APIs described at `https://jax-ws-architecture-document.dev.java.net/nonav/doc/com/sun/xml/ws/developer/package-summary.html`) are supported as an extension to the JDK 6.0 or JAX-WS 2.1 Reference Implementation (RI), provided by Sun Microsystems. Because the APIs are not provided as part of the JDK 6.0 or WebLogic Server software, they are subject to change. The APIs include, but are not limited to:

```
com.sun.xml.ws.api.server.AsyncProvider
com.sun.xml.ws.client.BindingProviderProperties
com.sun.xml.ws.developer.JAXWSProperties
com.sun.xml.ws.developer.SchemaValidation
com.sun.xml.ws.developer.SchemaValidationFeature
com.sun.xml.ws.developer.StreamingAttachment
com.sun.xml.ws.developer.StreamingAttachmentFeature
com.sun.xml.ws.developer.StreamingDataHandler
```

4.2 Apache XMLBeans 2.0

Apache XMLBeans 2.0, described at `http://xmlbeans.apache.org`, provides a technology for binding XML schema to Java types and for accessing XML data in a variety of ways. XMLBeans uses XML Schema to compile Java interfaces and classes that use to access and modify XML instance data. XMLBeans is the default binding technology for JAX-RPC Web services.

4.3 Java API for XML Registries (JAXR) 1.0

The *Java API for XML Registries (JAXR)* specification, described at `http://java.sun.com/webservices/jaxr`, provides a uniform and standard Java API for accessing different kinds of XML Registries. An XML registry is an enabling infrastructure for building, deploying, and discovering Web services.

Currently there are a variety of specifications for XML registries including, most notably, the ebXML Registry and Repository standard, which is being developed by OASIS and U.N./CEFACT, and the UDDI specification, which is being developed by a vendor consortium.

JAXR enables Java software programmers to use a single, easy-to-use abstraction API to access a variety of XML registries. Simplicity and ease of use are facilitated within JAXR by a unified JAXR information model, which describes content and metadata within XML registries.

4.4 Java API for XML-based RPC (JAX-RPC) 1.1

Namespace: `http://java.sun.com/xml/ns/jax-rpc`

The *Java API for XML-based RPC (JAX-RPC)* specification, described at `https://jax-rpc.dev.java.net/`, is a Sun Microsystems specification that defines the Java APIs for making XML-based remote procedure calls (RPC). In particular, these APIs are used to invoke and get a response from a Web service using SOAP 1.1, and XML-based protocol for exchange of information in a decentralized and distributed environment.

WebLogic Server implements all required features of the JAX-RPC Version 1.1 specification. Additionally, WebLogic Server implements optional data type support, as described in "Understanding Data Binding" in *Getting Started With WebLogic Web Services Using JAX-RPC*. WebLogic Server does not implement optional features of the JAX-RPC specification, other than what is described in this chapter.

4.5 Java API for XML-based Web Services (JAX-WS) 2.1

Namespace: `http://java.sun.com/xml/ns/jaxws`

The *Java API for XML-based Web Services (JAX-WS)* specification, described at `http://jcp.org/aboutJava/communityprocess/mrel/jsr224/index2.html`, is a standards-based API for coding, assembling, and deploying Java Web services. The "integrated stack" includes JAX-WS 2.1, Java Architecture for XML Binding (JAXB) 2.1 and SOAP with Attachments API for Java (SAAJ) 1.3. JAX-WS is designed to take the place of JAX-RPC in Web services and Web applications.

4.6 Java Architecture for XML Binding (JAXB) 2.1

Namespace: `http://java.sun.com/xml/ns/jaxb`

The *Java Architecture for XML Binding (JAXB)* specification, described at `http://jcp.org/en/jsr/detail?id=222`, provides a convenient way to bind an XML schema to a representation in Java code. This makes it easy for you to incorporate XML data and processing functions in applications based on Java technology without having to know much about XML itself.

Note: JAXB cannot be used with JAX-RPC.

4.7 JSR 109: Implementing Enterprise Web Services 1.2

The *JSR 109: Implementing Enterprise Web Services* specification, described at http://www.jcp.org/en/jsr/detail?id=109, defines the programming model and runtime architecture for implementing Web services in Java that run on a Java EE application server, such as WebLogic Server. In particular, it specifies that programmers implement Java EE Web services using one of two components:

- Java class running in the Web container
- Stateless session EJB running in the EJB container

The specification also describes a standard Java EE Web services packaging format, deployment model, and runtime services, all of which are implemented by WebLogic Web services.

4.8 Security Assertion Markup Language (SAML) 2.0 and 1.1

Namespaces:

`urn:oasis:names:tc:SAML:2.0:assertion`

`urn:oasis:names:tc:SAML:2.0:protocol`

The *Security Assertion Markup Language (SAML)* specification provides an XML standard for exchanging authentication and authorization data between security domains. For more information, see:

- http://www.oasis-open.org/specs/index.php#samlv2.0
- http://www.oasis-open.org/specs/index.php#samlv1.1

4.9 Security Assertion Markup Language (SAML) Token Profile 1.1 and 1.0

Namespace: `urn:oasis:names:tc:SAML:1.0:assertion`

The *Web Services Security: SAML Token Profile 1.1* specification defines a set of SOAP extensions that implement SOAP message authentication and encryption. For more information, see: .

- http://www.oasis-open.org/committees/download.php/16768/wss-v1.1-spec-os-SAMLTokenProfile.pdf
- http://docs.oasis-open.org/wss/oasis-wss-saml-token-profile-1.0.pdf

4.10 Simple Object Access Protocol (SOAP) 1.1 and 1.2

Namespace: http://schemas.xmlsoap.org/wsdl/soap

Simple Object Access Protocol (SOAP), described at http://www.w3.org/TR/SOAP, is a lightweight XML-based protocol used to exchange information in a decentralized, distributed environment. WebLogic Server includes its own implementation of versions 1.1 and 1.2 of the SOAP specification. The protocol consists of:

- An envelope that describes the SOAP message. The envelope contains the body of the message, identifies who should process it, and describes how to process it.
- A set of encoding rules for expressing instances of application-specific data types.
- A convention for representing remote procedure calls and responses.

This information is embedded in a Multipurpose Internet Mail Extensions (MIME)-encoded package that can be transmitted over HTTP, HTTPs, or other Web protocols. MIME is a specification for formatting non-ASCII messages so that they can be sent over the Internet.

The following example shows a SOAP 1.1 request for stock trading information embedded inside an HTTP request:

```
POST /StockQuote HTTP/1.1
Host: www.sample.com:7001
Content-Type: text/xml; charset="utf-8"
Content-Length: nnnn
SOAPAction: "Some-URI"

<SOAP-ENV:Envelope
   xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
         SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
     <SOAP-ENV:Body>
          <m:GetLastStockQuote xmlns:m="Some-URI">
               <symbol>ORCL</symbol>
          </m:GetLastStockQuote>
     </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

By default, WebLogic Web services use version 1.1 of SOAP; if you want your Web services to use version 1.2, you must specify the binding type in the JWS file that implements your service.

4.11 SOAP with Attachments API for Java (SAAJ) 1.3

The *SOAP with Attachments API for Java (SAAJ)* specification, described at `https://saaj.dev.java.net`, describes how developers can produce and consume messages conforming to the SOAP 1.1 specification and SOAP with Attachments notes.

The single package in the API, `javax.xml.soap`, provides the primary abstraction for SOAP messages with MIME attachments. Attachments may be entire XML documents, XML fragments, images, text documents, or any other content with a valid MIME type. In addition, the package provides a simple client-side view of a request-response style of interaction with a Web service.

4.12 Web Services Addressing (WS-Addressing) 1.0

Namespace: `http://www.w3.org/2005/08/addressing`

Note: In addition, the current release supports *Web Services Addressing (August 2004 Member Submission)*, described at `http://www.w3.org/Submission/2004/SUBM-ws-addressing-20040810`.

The *Web Services Addressing (WS-Addressing)* specification, described at `http://www.w3.org/TR/ws-addr-core`, provides transport-neutral mechanisms to address Web services and messages. In particular, the specification defines a number of XML elements used to identify Web service endpoints and to secure end-to-end endpoint identification in messages.

4.13 Web Services Atomic Transaction (WS-AtomicTransaction) Version 1.2, 1.1, and 1.0

The *Web Services Atomic Transaction (WS-AtomicTransaction)* specification, described at http://docs.oasis-open.org/ws-tx/wstx-wsat-1.2-spec-cs-01/wstx-wsat-1.2-spec-cs-01.html, defines the Atomic Transaction coordination type that is to be used with the extensible coordination framework described in the Web Services Coordination specification. The WS-AtomicTransaction and WS-Coordination specifications define an extensible framework for coordinating distributed activities among a set of participants.

4.14 Web Services Coordination (WS-Coordination) Version 1.2, 1.1, and 1.0

The *Web Services Coordination (WS-Coordination)* specification, described at http://docs.oasis-open.org/ws-tx/wstx-wscoor-1.2-spec-cs-01/wstx-wscoor-1.2-spec-cs-01.html, defines an extensible framework for providing protocols that coordinate the actions of distributed applications. The WS-AtomicTransaction and WS-Coordination specifications define an extensible framework for coordinating distributed activities among a set of participants.

4.15 Web Services Description Language (WSDL) 1.1

Namespace: http://schemas.xmlsoap.org/wsdl

Web Services Description Language (WSDL), described at http://www.w3.org/TR/wsdl, is an XML-based specification that describes a Web service. A WSDL document describes Web services operations, input and output parameters, and how a client application connects to the Web service.

Developers of WebLogic Web services do not need to create the WSDL files; you generate these files automatically as part of the WebLogic Web services development process.

The following example, for informational purposes only, shows a WSDL file that describes the stock trading Web services StockQuoteService that contains the method GetLastStockQuote:

```
<?xml version="1.0"?>
  <definitions name="StockQuote"
               targetNamespace="http://sample.com/stockquote.wsdl"
               xmlns:tns="http://sample.com/stockquote.wsdl"
               xmlns:xsd="http://www.w3.org/2000/10/XMLSchema"
               xmlns:xsd1="http://sample.com/stockquote.xsd"
               xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
               xmlns="http://schemas.xmlsoap.org/wsdl/">
      <message name="GetStockPriceInput">
          <part name="symbol" element="xsd:string"/>
      </message>
      <message name="GetStockPriceOutput">
          <part name="result" type="xsd:float"/>
      </message>
      <portType name="StockQuotePortType">
          <operation name="GetLastStockQuote">
             <input message="tns:GetStockPriceInput"/>
             <output message="tns:GetStockPriceOutput"/>
          </operation>
      </portType>
      <binding name="StockQuoteSoapBinding" type="tns:StockQuotePortType">
          <soap:binding style="rpc"
                        transport="http://schemas.xmlsoap.org/soap/http"/>
          <operation name="GetLastStockQuote">
             <soap:operation soapAction="http://sample.com/GetLastStockQuote"/>
             <input>
                 <soap:body use="encoded" namespace="http://sample.com/stockquote"

encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"/>
             </input>
             <output>
             <soap:body use="encoded" namespace="http://sample.com/stockquote"

encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"/>
             </output>
          </operation>>
      </binding>
      <service name="StockQuoteService">
          <documentation>My first service</documentation>
          <port name="StockQuotePort" binding="tns:StockQuoteSoapBinding">
             <soap:address location="http://sample.com/stockquote"/>
          </port>
      </service>
  </definitions>
```

The WSDL specification includes optional extension elements that specify different types of bindings that can be used when invoking the Web service. The WebLogic Web services runtime:

- Fully supports SOAP bindings, which means that if a WSDL file includes a SOAP binding, the WebLogic Web services will use SOAP as the format and protocol of the messages used to invoke the Web service.
- Ignores HTTP GET and POST bindings, which means that if a WSDL file includes this extension, the WebLogic Web services runtime skips over the element when parsing the WSDL.
- Partially supports MIME bindings, which means that if a WSDL file includes this extension, the WebLogic Web services runtime parses the element, but does not actually create MIME bindings when constructing a message due to a Web service invoke.

4.16 Web Services Metadata for the Java Platform 2.0 (JSR-181)

Oracle recommends that you take advantage of the metadata annotations feature, described at `http://java.sun.com/javase/6/docs/technotes/guides/language/annotations.html`, and use a programming model in which you create an annotated Java file and then use Ant tasks to convert the file into the Java source code of a standard Java class or EJB and automatically generate all the associated artifacts.

The Java Web Service (JWS) annotated file (called a *JWS file* for simplicity) is the core of your Web service. It contains the Java code that determines how your Web service behaves. A JWS file is an ordinary Java class file that uses JDK 5.0 metadata annotations to specify the shape and characteristics of the Web service. The JWS annotations you can use in a JWS file include the standard ones defined by the *Web Services Metadata for the Java Platform specification (JSR-181)*, described at `http://www.jcp.org/en/jsr/detail?id=181`, as well as a set of other standard or WebLogic-specific ones, depending on the type of Web service you are creating.

> **Note:** As an alternative to using a JWS annotated file, you can program a WebLogic Web service manually by coding the standard Java class or EJB from scratch and generating its associated artifacts by hand (deployment descriptor files, WSDL, data binding artifacts for user-defined data types, and so on). However, the entire process can be difficult and tedious and is not recommended.

4.17 Web Services Policy Attachment (WS-Policy Attachment) 1.5 and 1.2

Namespaces:

WS-Policy Attachment 1.5: `http://www.w3.org/ns/ws-policy`

WS-PolicyAttachment 1.2: `http://schemas.xmlsoap.org/ws/2004/09/policy`

The Web Services Policy Attachment (WS-Policy Attachment) specification defines an abstract model and an XML-based expression grammar for policies. The specification defines two general-purpose mechanisms for associating such policies with the subjects to which they apply. This specification also defines how these general-purpose mechanisms can be used to associate WS-Policy with WSDL and UDDI descriptions.

For more information, see:

- *Web Services Policy 1.5 - Attachment* (Recommendation): `http://www.w3.org/TR/ws-policy-attach/`
- *Web Services Policy 1.2 - Attachment (WS-PolicyAttachment)* (Member Submission): `http://www.w3.org/Submission/WS-PolicyAttachment`

4.18 Web Services Policy Framework (WS-Policy) 1.5 and 1.2

Namespaces:

WS-Policy Framework 1.5: `http://www.w3.org/ns/ws-policy`

WS-Policy 1.2: `http://schemas.xmlsoap.org/ws/2004/09/policy`

The WS-Policy Framework (WS-Policy) specification provides a general purpose model and corresponding syntax to describe and communicate the policies of a Web service. WS-Policy defines a base set of constructs that can be used and extended by other Web services specifications to describe a broad range of service requirements, preferences, and capabilities.

For more information, see:

- *Web Services Policy 1.5 - Framework* (Recommendation): `http://www.w3.org/TR/ws-policy`
- *Web Services Policy 1.2 - Framework (WS-Policy)* (Member Submission): `http://www.w3.org/Submission/WS-Policy`

4.19 Web Services Reliable Messaging (WS-ReliableMessaging) 1.1

Namespace: `http://docs.oasis-open.org/ws-rx/wsrm/200702`

The *Web Services Reliable Messaging (WS-ReliableMessaging)* specification, described at `http://docs.oasis-open.org/ws-rx/wsrm/200702/wsrm-1.1-spec-os-01.pdf`, describes how two Web services running on different WebLogic Server instances can communicate reliably in the presence of failures in software components, systems, or networks. In particular, the specification provides for an interoperable protocol in which a message sent from a source endpoint to a destination endpoint is guaranteed either to be delivered or to raise an error.

Note: The WS-ReliableMessaging 1.0 specification is supported for backward compatibility. However, a WS-ReliableMessaging 1.1 client cannot communicate with a WS-ReliableMessaging 1.0 server.

4.20 Web Services Reliable Messaging Policy Assertion (WS-RM Policy) 1.1

Namespace: `http://docs.oasis-open.org/ws-rx/wsrmp/200702`

The *Web Services Reliable Messaging Policy Assertion (WS-RM Policy) Version 1.1* specification, described at `http://docs.oasis-open.org/ws-rx/wsrmp/200702/wsrmp-1.1-spec-os-01.pdf`, defines a domain-specific policy assertion for reliable messaging for use with WS-Policy and WS-ReliableMessaging. This specification enables an RM Destination and an RM Source to describe their requirements for a given sequence.

4.21 Web Services Secure Conversation Language (WS-SecureConversation) 1.3

Namespace:
`http://docs.oasis-open.org/ws-sx/ws-secureconversation/200512`

The *Web Services Secure Conversation Language (WS-SecureConversation)* specification, described at `http://docs.oasis-open.org/ws-sx/ws-secureconversation/200512/ws-secureconversation-1.3-os.html`, defines extensions that build on Web Services Security (WS-Security) 1.1 and 1.0 and Web Services Trust Language (WS-Trust) 1.3 to provide secure communication across one or more messages. Specifically, this specification defines mechanisms for establishing and sharing security contexts, and deriving keys from established security contexts (or any shared secret).

4.22 Web Services Security (WS-Security) 1.1 and 1.0

Namespaces:
`http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecuritysecext-1.0.xsd`,
`http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurityutility-1.0.xsd`,
`http://docs.oasis-open.org/wss/oasis-wss-wssecurity-secext-1.1.xsd`

The following description of Web Services Security is taken directly from the OASIS standard 1.1 specification, titled *Web Services Security: SOAP Message Security*, dated February 2006:

This specification proposes a standard set of SOAP [SOAP11, SOAP12] extensions that can be used when building secure Web services to implement message content integrity and confidentiality. This specification refers to this set of extensions and modules as the "Web Services Security: SOAP Message Security" or "WSS: SOAP Message Security".

This specification is flexible and is designed to be used as the basis for securing Web services within a wide variety of security models including PKI, Kerberos, and SSL. Specifically, this *specification provides support for multiple security token formats, multiple trust domains, multiple signature formats, and multiple encryption technologies. The token formats and semantics for using these are defined in the associated profile documents.*

This specification provides three main mechanisms: ability to send security tokens as part of a message, message integrity, and message confidentiality. These mechanisms by themselves do not provide a complete security solution for Web services. Instead, this specification is a building block that can be used in conjunction with other Web service extensions and higher-level application-specific protocols to accommodate a wide variety of security models and security technologies.

These mechanisms can be used independently (for example, to pass a security token) or in a tightly coupled manner (for example, signing and encrypting a message or part of a message and providing a security token or token path associated with the keys used for signing and encryption).

WebLogic Web services also implement the following token profiles:

- Web Services Security: SOAP Message Security
- Web Services Security: Username Token Profile
- Web Services Security: X.509 Certificate Token Profile
- Web Services Security: SAML Token Profile

For more information, see the OASIS Web Service Security Web page at `http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=wss`.

4.23 Web Services Security Policy (WS-SecurityPolicy) 1.2

Namespace: `http://docs.oasis-open.org/ws-sx/ws-securitypolicy/200702`

The *Web Services Security Policy (WS-SecurityPolicy)* specification, described at `http://docs.oasis-open.org/ws-sx/ws-securitypolicy/200702/ws-securitypolicy-1.2-spec-os.html`, defines a set of security policy assertions for use with the WS-Policy framework to describe how messages are to be secured in the context of WS-Security, WS-Trust and WS-SecureConversation.

All the asynchronous features of WebLogic Web services (callbacks, conversations, and Web service reliable messaging) use addressing in their implementation, but Web service programmers can also use the APIs that conform to this specification stand-alone if additional addressing functionality is needed.

4.24 Web Services Trust Language (WS-Trust) 1.3

Namespace: `http://schemas.xmlsoap.org/ws/2005/02/trust`

The *Web Services Trust Language (WS-Trust)* specification, described at `http://specs.xmlsoap.org/ws/2005/02/trust/WS-Trust.pdf`, defines extensions that build on Web Services Security (WS-Security) 1.1 and 1.0 to provide a framework for requesting and issuing security tokens, and to broker trust relationships.

4.25 Universal Description, Discovery, and Integration (UDDI) 2.0

Note: This feature is deprecated.

Namespace: `urn:uddi-org:api_v2`

The Universal Description, Discovery, and Integration (UDDI) specification, described at `http://uddi.xml.org`, defines a standard for describing a Web service; registering a Web service in a well-known registry; and discovering other registered Web services.

4.26 Additional Specifications Supported by WebLogic Web Services

- *XML Schema Part 1: Structures* described at `http://www.w3.org/TR/xmlschema-1`
- *XML Schema Part 2: Data Types* described at `http://www.w3.org/TR/xmlschema-2`