Oracle® WebLogic Server

Introducing WebLogic Web Services

10*g* Release 3 (10.3)

July 2008

ORACLE

Oracle WebLogic Server Introducing WebLogic Web Services, 10*g* Release 3 (10.3)

Contents

4. Interoperability with Microsoft WCF/.NET

CHAPTER 1

Overview of WebLogic Web Services

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

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.

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.

Anatomy of a WebLogic Web Service

WebLogic Web Services are implemented according to the *Web Services for Java EE 1.2* specification, 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:

- The Programming Model—Metadata Annotations
- The Compilation Model—jwsc Ant Task

The Programming Model—Metadata Annotations

The *Web Services for Java EE 1.2* specification 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 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 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) and Java API for XML-Based RPC 1.1 (JAX-RPC) Web Services. 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 "How Do I Choose Between JAX-WS and JAX-RPC?" on page 1-5.

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). 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- 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 "Web Services Policy Framework (WS-Policy) 1.2".

The Compilation Model—jwsc Ant Task

After you create the JWS file, you use the `jwsc` WebLogic Web Service Ant task to compile the JWS file, as described by the Web Services for Java EE 1.2 specification, described in "Web Services for Java EE 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.

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 "Java API for XML-based Web Services (JAX-WS) 2.1" on page 3-5
- Java API for XML-Based RPC 1.1 (JAX-RPC), described in "Java API for XML-based RPC (JAX-RPC) 1.1" on page 3-4

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.

Note: JAX-WS Web Services do not support context propagation, as described in "Programming Context Propagation" in *Developing Applications With WebLogic Server*.

Table 1-1 Benefits of JAX-WS

Benefit	Description
SOAP 1.2 Support	JAX-WS supports SOAP 1.2 and 1.1. JAX-RPC supports SOAP 1.1 only.
Data Binding Using JAXB 2.1	JAX-WS 2.1 fully supports the Java Architecture for XML Binding (JAXB) 2.1 specification 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 WebLogic Web Services Using JAX-WS.* 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 "Using JAXB Data Binding" in *Getting Started With WebLogic Web Services Using JAX-RPC*.

Table 1-1 Benefits of JAX-WS (Continued)

Benefit	Description
Document Attachments Using MTOM	JAX-WS 2.1 supports MTOM (Message Transmission and Optimization Mechanism). MTOM, together with XOP (XML Binary Optimized Packaging) defines how an XML binary data such as `xs:base64Binary` or `xs:hexBinary` can be optimally transmitted over the wire. **Note:** In this release of WebLogic Server, MTOM is also supported for JAX-RPC 1.1 style Web Services.
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 *Web Services Metadata for the Java Platform (JSR 181)* specification 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*.
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.

Table 1-1 Benefits of JAX-WS (Continued)

Benefit	Description
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.

Roadmap for Implementing WebLogic Web Services

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

Table 1-2 Roadmap for Implementing WebLogic Web Services

Major Task	Subtasks and Additional Information
Review Supported Standards	"Standards Supported by WebLogic Web Services" on page 3-1
Run Samples	"Samples for WebLogic Web Service Developers" on page 2-1
	JAX-WS Use Cases and Examples
	JAX-RPC Use Cases and Examples

Table 1-2 Roadmap for Implementing WebLogic Web Services (Continued)

Major Task	Subtasks and Additional Information
Develop Web Services using JAX-WS	Getting Started With WebLogic Web Services Using JAX-WS
	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
	Publishing and Finding Web Services Using UDDI
Develop Web Services using JAX-RPC	Getting Started With WebLogic Web Services Using JAX-RPC
	Use Cases and Examples
	Invoking a Web Service Using Asynchronous Request-Response
	Using Web Service Reliable Messaging
	Creating Conversational Web Services
	Using the Asynchronous Features Together
	Using Callbacks to Notify Clients of Events
	Creating Buffered Web Services
	Using JMS Transport as the Connection Protocol
	Creating and Using SOAP Message Handlers
	Publishing and Finding Web Services Using UDDI

Table 1-2 Roadmap for Implementing WebLogic Web Services (Continued)

Major Task	Subtasks and Additional Information
Secure the Web Service	Configuring Message-Level Security (Digital Signatures and Encryption)
	Configuring Transport-Level Security
	Configuring Access Control Security
Upgrade	**JAX-WS**: No steps are required to upgrade JAX-WS to 10.3.
	JAX-RPC: Upgrading WebLogic Web Services From Previous Releases to 10.3

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 WebLogic Server" in *Release Notes*.

CHAPTER 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.

- Samples for WebLogic Web Service Developers
- WebLogic Web Services Documentation Set
- Related Documentation—WebLogic Server Application Development

Samples for WebLogic Web Service Developers

In addition to this document, Oracle provides a variety of code samples for Web Services developers. The examples 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 examples before programming your own application that use Web Services.

Web Services Examples in the WebLogic Server Distribution

WebLogic Server optionally installs API code examples 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 examples server, and obtain information about the samples and how to run them from the WebLogic Server Start menu.

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.

As companion documentation to the MedRec application, Oracle provides development tutorials that provide step-by-step procedures for key development tasks, including Web Service-specific tasks. See Sample Application and Code Examples for Oracle WebLogic Server 10g Release 3 for the latest information.

Additional Web Services Examples Available for Download

Additional API examples for download can be found at http://www.oracle.com/technology/index.html. These examples include Oracle-certified ones, as well as examples submitted by fellow developers.

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 (This Document)	An introduction to WebLogic Web Services, the standards that are supported, interoperability information, and relevant samples and documentation.
Getting Started With WebLogic Web Services Using JAX-WS	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 WebLogic Web Services Using JAX-WS	How to program more advanced features using JAX-WS, such as callbacks, XML Catalog, and SOAP message handlers.
Getting Started With WebLogic Web Services Using JAX-RPC	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 WebLogic Web Services Using JAX-RPC	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	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	Reference information on JWS annotations, Ant tasks, reliable messaging WS-Policy assertions, security WS-Policy assertions, and deployment descriptors.

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 . . .
Developing Applications With WebLogic Server	Develop WebLogic Server components (such as Web applications and EJBs) and applications.
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	Develop EJBs that are deployed and run on WebLogic Server.
Programming WebLogic XML	Design and develop applications that include XML processing.
Deploying Applications to 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 WebLogic Server*	Configure your applications for deployment to a production WebLogic Server environment.
Performance and Tuning	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.

CHAPTER 3

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 Services platforms to be more important than providing support for all possible edge cases of the Web Services specifications. Oracle complies with the Basic Profile 1.1 and Basic Security Profile 1.0 specifications from the Web Services Interoperability Organization and considers them to be the baseline for Web Services interoperability. This guide does not necessarily document all of the Basic Profile 1.1 and Basic Security Profile 1.0 requirements. This guide does, however, document features that are beyond the requirements of the Basic Profile 1.1 and Basic Security Profile 1.0.

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

Table 3-1 Oracle Implementation of Web Service Specifications

Feature	Specification	Description
Programming model (based on metadata annotations) and runtime architecture	Web Services for Java EE 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.
	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.

Table 3-1 Oracle Implementation of Web Service Specifications (Continued)

Feature	Specification	Description
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.
	Java API for XML-based RPC (JAX-RPC) 1.1	Java APIs for making XML-based remote procedure calls (RPC).
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.
	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.
Web Service description	Web Services Description Language (WSDL) 1.1	XML-based specification that describes a Web Service
	Web Services Policy Framework (WS-Policy) 1.2	General purpose model and corresponding syntax to describe and communicate the policies of a Web Service
	Web Services Policy Attachment (WS-PolicyAttachment) 1.2	Abstract model and an XML-based expression grammar for policies.
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
	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.
Security	Web Services Security (WS-Security) 1.1	Standard set of SOAP [SOAP11, SOAP12] extensions that can be used when building secure Web services to implement message content integrity and confidentiality.
	Web Services Security Policy (WS-SecurityPolicy) 1.2	Set of security policy assertions for use with the WS-Policy framework.

Table 3-1 Oracle Implementation of Web Service Specifications (Continued)

Feature	Specification	Description
Asynchronous communication	Web Services Addressing (WS-Addressing) 1.0	Transport-neutral mechanisms to address Web services and messages.
	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.
	Web Services Reliable Messaging Policy (WS-ReliableMessaging Policy) 1.1	Domain-specific policy assertion for reliable messaging for use with WS-Policy and WS-ReliableMessaging.
	Web Services Trust Language (WS-Trust) 1.3	Extensions that build on Web Services Security (WS-Security) 1.1 to secure asynchronous 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.
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.
	Java API for XML Registries (JAX-R) 1.0	Uniform and standard Java API for accessing different kinds of XML Registries.

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 "Additional Specifications Supported by WebLogic Web Services" on page 3-12.

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) 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
```

Apache XMLBeans 2.0

Apache XMLBeans 2.0 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.

Java API for XML Registries (JAX-R) 1.0

The Java API for XML Registries (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.

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

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

Java API for XML-based RPC (JAX-RPC) 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.

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

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

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

Java Architecture for XML Binding (JAXB) 2.1

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

Java Architecture for XML Binding (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.

Note: You can use JAXB with JAX-WS Web Services *only*.

Simple Object Access Protocol (SOAP) 1.1 and 1.2

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

Simple Object Access Protocol (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 Service to use version 1.2, you must specify the binding type in the JWS file that implements your service.

SOAP with Attachments API for Java (SAAJ) 1.3

The SOAP with Attachments API for Java (SAAJ) specification 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.

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).

The Web Services Addressing (WS-Addressing) specification 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.

Web Services Description Language (WSDL) 1.1

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

Web Services Description Language (WSDL) is an XML-based specification that describes a Web Service. A WSDL document describes Web Service 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 Service 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.

Web Services for Java EE 1.2

The Web Services for Java EE 1.2 specification (JSR-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 Service packaging format, deployment model, and runtime services, all of which are implemented by WebLogic Web Services.

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

Oracle recommends that you take advantage of the new JDK 5.0 metadata annotations feature 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) as well as a set of other standard or WebLogic-specific ones, depending 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.

Web Services Policy Attachment (WS-PolicyAttachment) 1.2

Namespace: `http://schemas.xmlsoap.org/ws/2004/09/policy`

The *Web Services Policy Framework (WS-Policy)* specification defines an abstract model and an XML-based expression grammar for policies. This specification, Web Services Policy Attachment (WS-PolicyAttachment), 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.

Web Services Policy Framework (WS-Policy) 1.2

Namespace: `http://schemas.xmlsoap.org/ws/2004/09/policy`

Note: In addition, the current release supports W3C Web Services Policy Framework (WS-Policy) 1.5 (namespace: `http://www.w3.org/ns/ws-policy`).

The WS-Policy 1.2 (Member Submission) 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.

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 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.

Web Services Reliable Messaging Policy (WS-ReliableMessaging Policy) 1.1

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

The Web Services Reliable Messaging Policy (WS-ReliableMessaging Policy) specification 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.

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 defines extensions that build on Web Services Security (WS-Security) 1.1 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).

Web Services Security (WS-Security) 1.1

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 1.1

For more information, see the OASIS Web Service Security Web page.

Web Services Security Policy (WS-SecurityPolicy) 1.2

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

Web Services Security Policy (WS-SecurityPolicy) 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.

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 defines extensions that build on Web Services Security (WS-Security) 1.1 to provide a framework for requesting and issuing security tokens, and to broker trust relationships.

Universal Description, Discovery, and Integration (UDDI) 2.0

Namespace: `urn:uddi-org:api_v2`

The Universal Description, Discovery, and Integration (UDDI) specification defines a standard for describing a Web Service; registering a Web Service in a well-known registry; and discovering other registered Web Services.

Additional Specifications Supported by WebLogic Web Services

- XML Schema Part 1: Structures

- XML Schema Part 2: Data Types

CHAPTER 4

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.

Interoperability tests were completed on JAX-RPC Web Services in the following areas:

Table 4-1 Completed Interoperability Tests

Area	Interoperability Guidelines
Basic and complex data types	"Basic Data Types Interoperability Guidelines" on page 4-2.
Web Services Security (WS-Security) 1.0 and 1.1	"WS-Security Interoperability Guidelines" on page 4-2.
Web Services Security Policy (WS-SecurityPolicy) 1.2	"WS-SecurityPolicy Interoperability Guidelines" on page 4-3
Web Services Secure Conversation Language (WS-SecureConversation) 1.3	"WS-SecureConversation Interoperability Guidelines" on page 4-3
Web Services Policy Framework (WS-Policy) 1.5	No interoperability guidelines provided.
Web Services Addressing (WS-Addressing) 0.9 and 1.0	N/A

Table 4-1 Completed Interoperability Tests (Continued)

Area	Interoperability Guidelines
Message Transmission Optimization Mechanism (MTOM)	N/A
Web Services Reliable Messaging (WS-ReliableMessaging) 1.0 and 1.1	"WS-ReliableMessaging Interoperability Guidelines" on page 4-4
Web Services Trust (WS-Trust) 1.3	"WS-Trust Interoperability Guidelines" on page 4-4

In addition, the following combined features were tested:

- MTOM and WS-Security
- WS-ReliableMessaging and MTOM
- WS-ReliableMessaging 1.0 and WS-Addressing 0.9 and 1.0
- WS-ReliableMessaging 1.1 and WS-Addressing 0.9 and 1.0
- WS-ReliableMessaging 1.0 and WS-SecureConversation 1.3
- WS-ReliableMessaging 1.0 and WS-SecureConversation 1.3
- WS-Policy 1.5 and WS-SecurityPolicy 1.2

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

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`.

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 .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.

WS-SecurityPolicy Interoperability Guidelines

In this release, WebLogic Server supports 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 .NET Web Services will not interoperate.

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 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 "WS-Security Interoperability Guidelines" on page 4-2.
- 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 .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");
    ```

WS-ReliableMessaging Interoperability Guidelines

The following lists interoperability guidelines for WS-ReliableMessaging:

- Anonymous request/response is not defined in the WS-ReliableMessaging specification and is, consequently, not supported by WebLogic Server.
- For WS-ReliableMessaging security, you must use WS-SecureConversation.
- Asynchronous reliable messaging plus WS-SecureConversation or WS-Trust is not supported.

WS-Trust Interoperability Guidelines

WebLogic Server does not interoperate with Microsoft .NET according to the .NET interoperability scenarios that use both SAML and WS-Trust, as defined in the following documents on the Microsoft .NET Web Services Interoperability site:

- WS-SX Scenarios Document
- WCF (Indigo) Interoperability Lab: WS-Trust10 Scenarios