Oracle® Fusion Middleware

Understanding WebLogic Web Services for Oracle WebLogic Server

14c (14.1.2.0.0)
F53013-01
December 2024

Oracle Fusion Middleware Understanding WebLogic Web Services for Oracle WebLogic Server, 14c (14.1.2.0.0)

F53013-01

Primary Author: Oracle Corporation

Contents

6 Examples for Jakarta EE Web Service Developers

Preface

This documentation introduces web services for Oracle WebLogic Server 14c, including interoperability and standards information.

Audience

This document is a resource for software developers who develop Jakarta EE Web Services for Oracle WebLogic Server 14c using the Java API for XML-based Web services (JAX-WS). It is assumed that the reader is familiar with Jakarta EE and JAX-WS concepts.

Documentation Accessibility

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

Access to Oracle Support

Oracle customer access to and use of Oracle support services will be pursuant to the terms and conditions specified in their Oracle order for the applicable services.

Diversity and Inclusion

Oracle is fully committed to diversity and inclusion. Oracle respects and values having a diverse workforce that increases thought leadership and innovation. As part of our initiative to build a more inclusive culture that positively impacts our employees, customers, and partners, we are working to remove insensitive terms from our products and documentation. We are also mindful of the necessity to maintain compatibility with our customers' existing technologies and the need to ensure continuity of service as Oracle's offerings and industry standards evolve. Because of these technical constraints, our effort to remove insensitive terms is ongoing and will take time and external cooperation.

Related Documentation

New and Changed WebLogic Server Features

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

Conventions

The following text conventions are used in this document:

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

1

Introducing Oracle WebLogic Web Services

WebLogic Web services for Oracle WebLogic Server are loosely coupled, distributed environments that allow you to integrate heterogeneous applications within the enterprise or to expose business functions to customers and partners over the Internet. These services are characterized by the business functionality, the website which exposes that functionality, and the set of published interfaces necessary to use the exposed functionality.
For definitions of unfamiliar terms found in this and other books, see the Glossary.

- Overview of WebLogic Web Services
  You can access the Web services using standard Web protocols such as XML or HTTP. WebLogic Server supports the web service types such as Java API for XML-Based Web Services (JAX-WS) 2.3 and Java API for RESTful Web Services (JAX-RS).
- How Do I Choose Between SOAP and REST?
  In WebLogic Server, SOAP web services are implemented using JAX-WS and RESTful web services are implemented using JAX-RS. Follow the recommended guidelines to consider when choosing between SOAP and REST.

Overview of WebLogic Web Services

You can access the Web services using standard Web protocols such as XML or HTTP. WebLogic Server supports the web service types such as Java API for XML-Based Web Services (JAX-WS) 2.3 and Java API for RESTful Web Services (JAX-RS).

For an overview of web services and their benefits, see What Are Web Services? in *Understanding Web Services*.

Table 1-1 Types of WebLogic Web Services

Web Service Type	Description
Java API for XML-Based Web Services (JAX-WS) 2.3	The JAX-WS implementation in Oracle WebLogic Server is extended from the JAX-WS Reference Implementation (RI) developed by the Glassfish Community (see https://github.com/eclipse-ee4j/metro-jax-ws). For more information about JAX-WS, see: • *Developing JAX-WS Web Services for Oracle WebLogic Server* • JAX-WS specification: https://www.jcp.org/en/jsr/detail?id=224
Java API for RESTful Web Services (JAX-RS)	WebLogic Server supports Jersey 2.x (JAX-RS 2.1 RI) by default in this release. Registration as a shared library with WebLogic Server is no longer required. For more information about JAX-RS, see: • *Developing and Securing RESTful Web Services for Oracle WebLogic Server* • JAX-RS 2.1 specification: https://jcp.org/en/jsr/detail?id=370

How Do I Choose Between SOAP and REST?

In WebLogic Server, SOAP web services are implemented using JAX-WS and RESTful web services are implemented using JAX-RS. Follow the recommended guidelines to consider when choosing between SOAP and REST.

See also Features and Standards Supported by WebLogic Web Services for a comparison of the standards that are supported for JAX-WS and JAX-RS.

Table 1-2 How to Choose Between SOAP and RESTful Web Services

Use . . .	In the following scenarios . . .
SOAP	Implement SOAP web services using JAX-WS in enterprise application integration scenarios that: • Have advanced quality of service (QoS) requirements. • Need to call methods remotely in Java components, such as Plain Old Java Objects (POJOs) or Jakarta Enterprise Beans (EJBs). JAX-WS interoperates with other standards-based SOAP web services from Oracle or other SOAP web service vendors. JAX-WS supports the full set of WS-* protocols that provide standards for security, reliability, and so on, and better interoperates with other clients and servers that conform to the WS-* protocols. For more information about SOAP web service development, see *Developing JAX-WS Web Services for Oracle WebLogic Server*.
REST	Implement RESTful web services using JAX-RS to integrate services over the web when the constraints of the RESTful style are desirable, such as separate client-server architecture, uniform interface, and so on. For more information about RESTful web services development, see *Developing and Securing RESTful Web Services for Oracle WebLogic Server*.

2

Features and Standards Supported by WebLogic Web Services

WebLogic web services for Oracle WebLogic Server support various features and standards. Many specifications that define web service standards are written to allow for broad use of the specification throughout the industry. The Oracle implementation of a particular specification may not cover all possible usage scenarios defined in the specifications.

> **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://github.com/eclipse-ee4j/metro-jax-ws). 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.

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 2.0* (JAX-WS only): http://docs.oasis-open.org/ws-brsp/BasicProfile/v2.0/BasicProfile-v2.0.html
- *Basic Profile Version 1.2* (JAX-WS only): http://docs.oasis-open.org/ws-brsp/BasicProfile/v1.2/BasicProfile-v1.2.html
- *Basic Profile Version 1.1* (JAX-WS only): http://www.ws-i.org/Profiles/BasicProfile-1.1-2004-08-24.html
- *Basic Security Profile 1.1* (JAX-WS only): http://docs.oasis-open.org/ws-brsp/BasicSecurityProfile/v1.1/BasicSecurityProfile-v1.1.html
- *Reliable Secure Profile Version 1.0* (JAX-WS only): http://docs.oasis-open.org/ws-brsp/ReliableSecureProfile/v1.0/ReliableSecureProfile-v1.0.html

The WebLogic web service documentation set does not necessarily document all of the specification requirements; it does, however, document features that are beyond the requirements of these specifications.

The following table summarizes the features and specifications supported by WebLogic web services.

Table 2-1 Features and Standards Supported by WebLogic Web Services

Feature	Description	JAX-WS	JAX-RS
Programming model (based on metadata annotations) and runtime architecture	**JSR 109: Implementing Enterprise Web Services**—Programming model and runtime architecture for implementing web services in Java that run on a Jakarta EE application server, such as WebLogic Server. See JSR 109: Implementing Enterprise Web Services 1.4.	Version 1.4	N/A
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 Web Services Metadata for the Java Platform 2.1 (JSR-181).	Supports	N/A
Programming APIs	**Java API for XML-based Web Services (JAX-WS)**—Standards-based API for coding, assembling, and deploying Java web services. The integrated stack includes JAX-WS 2.3, JAXB 2.3, and SAAJ 1.3. See Java API for XML-based Web Services (JAX-WS) 2.3. See also *Developing JAX-WS Web Services for Oracle WebLogic Server*.	Version 2.3	N/A
Programming APIs	**Java API for RESTful Web Services (JAX-RS)**—Provides a standard JAVA API for developing web services based on the Representational State Transfer (REST) architectural style. See Java API for RESTful Web Services (JAX-RS). See also *Developing and Securing RESTful Web Services for Oracle WebLogic Server*.	N/A	2.1
Data binding	**Java Architecture for XML Binding (JAXB)**—Implementation used to bind an XML schema to a representation in Java code. JAXB is supported by JAX-WS web services only. See Java Architecture for XML Binding (JAXB) 2.3. See also Using JAXB Data Binding in *Developing JAX-WS Web Services for Oracle WebLogic Server*.	Version 2.3	Version 2.3
Web service description	**Web Services Description Language (WSDL)**—XML-based specification that describes a web service. See Web Services Description Language (WSDL) 1.1. See also Developing WebLogic Web Services Starting from a WSDL File: Main Steps in *Developing JAX-WS Web Services for Oracle WebLogic Server*.	Version 1.1	N/A
Web service description	**Web Application Description Language (WADL)**—XML-based specification that provides a machine-readable description of HTTP-based Web applications. See Web Application Description Language (WADL) 2009 Membership Submission.	N/A	2009 Member Submission
Web service description	**Web Services Policy Framework (WS-Policy)**—General purpose model and corresponding syntax to describe and communicate the policies of a web service. See Web Services Policy Framework (WS-Policy) 1.5 and 1.2.	Versions 1.5 and 1.2	N/A

Table 2-1 (Cont.) Features and Standards Supported by WebLogic Web Services

Feature	Description	JAX-WS	JAX-RS
Web service description	**Web Services Policy Attachment (WS-PolicyAttachment)**—Abstract model and an XML-based expression grammar for policies. See Web Services Policy Attachment (WS-Policy Attachment) 1.5 and 1.2.	Versions 1.5 and 1.2	N/A
Data exchange between web service and requesting client	**Simple Object Access Protocol (SOAP)**—Lightweight XML-based protocol used to exchange information in a decentralized, distributed environment. See Simple Object Access Protocol (SOAP) 1.1 and 1.2.	Versions 1.2 and 1.1	N/A
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 SOAP with Attachments API for Java (SAAJ) 1.3.	Version 1.3	N/A
Security	**Web Services Security (WS-Security)**—Standard set of SOAP [SOAP11, SOAP12] extensions that can be used when building secure web services to implement message content integrity and confidentiality. See Web Services Security (WS-Security) 1.1 and 1.0. See also *Securing WebLogic Web Services for Oracle WebLogic Server*.	Versions 1.1 and 1.0	N/A
Security	**Web Services Security Policy (WS-SecurityPolicy)**—Set of security policy assertions for use with the WS-Policy framework. See Web Services Security Policy (WS-SecurityPolicy) 1.3. See also *Securing WebLogic Web Services for Oracle WebLogic Server*.	Version 1.3	N/A
Security	**Security Assertion Markup Language (SAML)**—XML standard for exchanging authentication and authorization data between security domains. See Security Assertion Markup Language (SAML) 2.0 and 1.1. See also *Securing WebLogic Web Services for Oracle WebLogic Server*.	Versions 2.0 and 1.1	N/A
Security	**Security Assertion Markup Language (SAML) Token Profile**—Set of WS-Security SOAP extensions that implement SOAP message authentication and encryption. See Security Assertion Markup Language (SAML) Token Profile 1.1 and 1.0. See also *Securing WebLogic Web Services for Oracle WebLogic Server*.	Versions 1.1 and 1.0	N/A
Reliable communication	**Web Services Addressing (WS-Addressing)**—Transport-neutral mechanisms to address web services and messages. See Web Services Addressing (WS-Addressing) 1.0 and 2004/08 Member Submission.	Version 1.0 and 2004/08	N/A

Table 2-1 (Cont.) Features and Standards Supported by WebLogic Web Services

Feature	Description	JAX-WS	JAX-RS
Reliable communication	**Web Services Reliable Messaging (WS-ReliableMessaging)**—Implementation that enables two endpoints (web service and client) running on different WebLogic Server instances to communicate reliably in the presence of failures in software components, systems, or networks. See Web Services Reliable Messaging (WS-ReliableMessaging). See also Using Web Services Reliable Messaging in *Developing JAX-WS Web Services for Oracle WebLogic Server*.	Versions 1.2, 1.1	N/A
Reliable communication	**Web Services Reliable Messaging Policy Assertion (WS-RM Policy)**—Domain-specific policy assertion for reliable messaging for use with WS-Policy and WS-ReliableMessaging. See Web Services Reliable Messaging Policy Assertion (WS-RM Policy). See also Pre-packaged WS-Policy Files for Reliable Messaging and MakeConnection in *Developing JAX-WS Web Services for Oracle WebLogic Server*.	Versions 1.2 and 1.1	N/A
Reliable communication	**Web Services Trust Language (WS-Trust)**—Extensions that build on *Web Services Security (WS-Security)* to secure asynchronous communication. See Web Services Trust Language (WS-Trust). See also Configuring Message-Level Security in *Securing WebLogic Web Services for Oracle WebLogic Server*.	Version 1.4 and 1.3	N/A
Reliable communication	**Web Services Secure Conversation Language (WS-SecureConversation)**—Extensions that build on *Web Services Security (WS-Security)* and *Web Services Trust Language (WS-Trust)* to secure asynchronous communication. See Web Services Secure Conversation Language (WS-SecureConversation). See also Configuring Message-Level Security in *Securing WebLogic Web Services for Oracle WebLogic Server*.	Version 1.4	N/A
Asynchronous communication	**Asynchronous Request Response**—When you invoke a web service synchronously, the invoking client application waits for the response to return before it can continue with its work. In cases where the response returns immediately, this method of invoking the web service is common. However, because request processing can be delayed, it is often useful for the client application to continue its work and handle the response later on. This can be accomplished using asynchronous web service invocation. For example, see Developing Asynchronous Clients in *Developing JAX-WS Web Services for Oracle WebLogic Server*.	Supported	Supported

Table 2-1 (Cont.) Features and Standards Supported by WebLogic Web Services

Feature	Description	JAX-WS	JAX-RS
Asynchronous communication	**WS-MakeConnection**—Provides a mechanism for the transfer of messages between two endpoints when the sending endpoint is unable to initiate a new connection to the receiving endpoint. See Web Services MakeConnection 1.1. See also Developing Asynchronous Clients in *Developing JAX-WS Web Services for Oracle WebLogic Server*.	Version 1.1	N/A
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 Web Services Atomic Transaction (WS-AtomicTransaction) Version 1.2, 1.1, and 1.0. See also Using Web Services Atomic Transactions in *Developing JAX-WS Web Services for Oracle WebLogic Server*.	Versions 1.2, 1.1, and 1.0	N/A
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 Web Services Coordination (WS-Coordination) Version 1.2, 1.1, and 1.0. See also Using Web Services Atomic Transactions in *Developing JAX-WS Web Services for Oracle WebLogic Server*.	Versions 1.2, 1.1, and 1.0	N/A
Client event notification	**Web service callbacks**—Callbacks notify a client of your web service that some event has occurred. For example, you can notify a client when the results of that client's request are ready, or when the client's request cannot be fulfilled. For more information, see Using Callbacks in *Developing JAX-WS Web Services for Oracle WebLogic Server*.	Supported	Not supported
Optimizing XML transmission	**Fast Infoset**—Compressed binary encoding format that provides a more efficient serialization than the text-based XML format. Fast Infoset optimizes both document size and processing performance. See Fast Infoset. See also Optimizing XML Transmission Using Fast Infoset in *Developing JAX-WS Web Services for Oracle WebLogic Server*.	Supported	Not supported

Table 2-1 (Cont.) Features and Standards Supported by WebLogic Web Services

Feature	Description	JAX-WS	JAX-RS
Optimizing XML transmission	**Message Transmission Optimization Mechanism** (MTOM)—Defines a method for optimizing the transmission of XML data of type `xs:base64Binary` or `xs:hexBinary` in SOAP messages. For more information, see Optimizing Binary Data Transmission in *Developing JAX-WS Web Services for Oracle WebLogic Server*.	Supported	Not supported
SOAP Over JMS Transport	**SOAP over JMS transport**—Typically, client applications use HTTP/S as the connection protocol when invoking a WebLogic web service. You can, however, configure a WebLogic web service so that client applications use JMS as the transport instead. See SOAP Over JMS Transport 1.0. For more information, see Using JMS Transport as the Connection Protocol in *Developing JAX-WS Web Services for Oracle WebLogic Server*.	Supported	Not supported
Stand-alone Java SE client access	**Stand-alone Java SE client JAR file**—If your computer does not have WebLogic Server installed, you can still invoke a web service by using the stand-alone WebLogic web services client JAR file. See Invoking a Web Service from a Standalone Java SE Client in *Developing JAX-WS Web Services for Oracle WebLogic Server*.	Supported	Supported

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.

- A Note About JAX-WS 2.3 RI/JDK 17 Extensions
  A subset of the APIs such as `com.sun.xml.ws.developer` are supported as an extension to the JDK 17 or JAX-WS 2.3 Reference Implementation (RI).
- Fast Infoset
  Fast Infoset is a compressed binary encoding format that provides a more efficient serialization than the text-based XML format. Fast Infoset optimizes both document size and processing performance.
- Java API for RESTful Web Services (JAX-RS)
- Java API for XML-based Web Services (JAX-WS) 2.3
  The *Java API for XML-based Web Services (JAX-WS)* is a standards-based API for coding, assembling, and deploying Java web services.
- Java Architecture for XML Binding (JAXB) 2.3
  The *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.
- JSR 109: Implementing Enterprise Web Services 1.4
  The *JSR 109: Implementing Enterprise Web Services* defines the programming model and runtime architecture for implementing web services in Java that run on a Jakarta EE application server, such as WebLogic Server.

- Security Assertion Markup Language (SAML) 2.0 and 1.1
  The *Security Assertion Markup Language (SAML)* specification provides an XML standard for exchanging authentication and authorization data between security domains.
- Security Assertion Markup Language (SAML) Token Profile 1.1 and 1.0
  The *Web Services Security: SAML Token Profile 1.1* specification defines a set of SOAP extensions that implement SOAP message authentication and encryption.
- Simple Object Access Protocol (SOAP) 1.1 and 1.2
  Simple Object Access Protocol (SOAP) is a lightweight XML-based protocol used to exchange information in a decentralized, distributed environment.
- SOAP Over JMS Transport 1.0
  SOAP over JMS services transport is supported as a connection protocol for JAX-WS WebLogic web services.
- SOAP with Attachments API for Java (SAAJ) 1.3
  The *SOAP with Attachments API for Java (SAAJ)* describes how developers can produce and consume messages conforming to the SOAP 1.1 specification and SOAP with Attachments notes.
- Web Application Description Language (WADL) 2009 Membership Submission
  Web Application Description Language (WADL) is an XML-based specification that provides a machine-readable description of HTTP-based Web applications. Developers of WebLogic web services do not need to create the WADL files; you generate these files automatically as part of the WebLogic web services development process.
- Web Services Addressing (WS-Addressing) 1.0 and 2004/08 Member Submission
  The *Web Services Addressing (WS-Addressing) Core* provides transport-neutral mechanisms to address web services and messages.
- Web Services Atomic Transaction (WS-AtomicTransaction) Version 1.2, 1.1, and 1.0
  The *Web Services Atomic Transaction (WS-AtomicTransaction)* 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.
- Web Services Coordination (WS-Coordination) Version 1.2, 1.1, and 1.0
  The *Web Services Coordination (WS-Coordination)* defines an extensible framework for providing protocols that coordinate the actions of distributed applications.
- Web Services Description Language (WSDL) 1.1
  Web Services Description Language (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.
- Web Services MakeConnection 1.1
  The *Web Services MakeConnection* provides a mechanism for the transfer of messages between two endpoints when the sending endpoint is unable to initiate a new connection to the receiving endpoint. For example, to enable asynchronous web service invocation from behind a firewall.
- Web Services Metadata for the Java Platform 2.1 (JSR-181)
  Oracle recommends that you take advantage of the metadata annotations feature in Oracle WebLogic Server. To do so, you 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.
- Web Services Policy Attachment (WS-Policy Attachment) 1.5 and 1.2
  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.

- Web Services Policy Framework (WS-Policy) 1.5 and 1.2
  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.
- Web Services Reliable Messaging (WS-ReliableMessaging)
  The *Web Services Reliable Messaging (WS-ReliableMessaging)* 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.
- Web Services Reliable Messaging Policy Assertion (WS-RM Policy)
  The *Web Services Reliable Messaging Policy Assertion (WS-RM 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)
  The Web Services Secure Conversation Language (WS-SecureConversation) specification defines extensions that build on Web Services Security (WS-Security) 1.1 and 1.0 and Web Services Trust Language (WS-Trust) to provide secure communication across one or more messages.
- Web Services Security (WS-Security) 1.1 and 1.0
- Web Services Security Policy (WS-SecurityPolicy) 1.3
  The *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.
- Web Services Trust Language (WS-Trust)
  The *Web Services Trust Language (WS-Trust)* defines extensions that provides a framework for requesting and issuing security tokens, and to broker trust relationships.
- Additional Specifications Supported by WebLogic Web Services

A Note About JAX-WS 2.3 RI/JDK 17 Extensions

A subset of the APIs such as `com.sun.xml.ws.developer` are supported as an extension to the JDK 17 or JAX-WS 2.3 Reference Implementation (RI).

Because the APIs are not provided as part of the JDK 17 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
```

Fast Infoset

Fast Infoset is a compressed binary encoding format that provides a more efficient serialization than the text-based XML format. Fast Infoset optimizes both document size and processing performance.

When enabled, Fast Infoset converts the XML Information Set in the SOAP envelope into a compressed binary format before transmitting the data. Fast Infoset optimizes encrypted and signed messages, MTOM-enabled messages, and SOAP attachments, and supports both HTTP and JMS transports.

The Fast Infoset specification, *ITU-T Rec. X.891 and ISO/IEC 24824-1 (Fast Infoset)* is defined by both the ITU-T and ISO standards bodies. The specification can be downloaded from the ITU Web site: `http://www.itu.int/rec/T-REC-X.891-200505-I/en`

See Optimizing XML Transmission Using Fast Infoset in *Developing JAX-WS Web Services for Oracle WebLogic Server*.

Java API for RESTful Web Services (JAX-RS)

The *Java API for RESTful Web Services (JAX-RS)* specification provides a standard JAVA API for developing web services based on the Representational State Transfer (REST) architectural style. See `https://jcp.org/en/jsr/detail?id=370`.
WebLogic Server provides support for Jersey 2.x (JAX-RS 2.1 RI) by default in this release. Registration as a shared library with WebLogic Server is no longer required.

See *Developing and Securing RESTful Web Services for Oracle WebLogic Server*.

Java API for XML-based Web Services (JAX-WS) 2.3

The *Java API for XML-based Web Services (JAX-WS)* is a standards-based API for coding, assembling, and deploying Java web services.

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

See `http://jcp.org/aboutJava/communityprocess/mrel/jsr224/index5.html`. The integrated stack includes JAX-WS 2.3, Java Architecture for XML Binding (JAXB) 2.3 and SOAP with Attachments API for Java (SAAJ) 1.3.

See *Developing JAX-WS Web Services for Oracle WebLogic Server*.

Java Architecture for XML Binding (JAXB) 2.3

The *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.

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

See `https://jcp.org/aboutJava/communityprocess/mrel/jsr222/index3.html`.

See Using JAXB Data Binding in *Developing JAX-WS Web Services for Oracle WebLogic Server*.

JSR 109: Implementing Enterprise Web Services 1.4

The *JSR 109: Implementing Enterprise Web Services* defines the programming model and runtime architecture for implementing web services in Java that run on a Jakarta EE application server, such as WebLogic Server.

See the *JSR 109: Implementing Enterprise Web Services* specification at http://www.jcp.org/en/jsr/detail?id=109. In particular, it specifies that programmers implement Jakarta 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 Jakarta EE web services packaging format, deployment model, and runtime services, all of which are implemented by WebLogic web services.

Security Assertion Markup Language (SAML) 2.0 and 1.1

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

Namespaces:

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

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

See:

- https://www.oasis-open.org/standards#samlv2.0
- https://www.oasis-open.org/standards#samlv1.1

See *Securing WebLogic Web Services for Oracle WebLogic Server*.

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

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

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

See:

- https://groups.oasis-open.org/higherlogic/ws/public/document?document_id=16768
- http://docs.oasis-open.org/wss/oasis-wss-saml-token-profile-1.0.pdf

See *Securing WebLogic Web Services for Oracle WebLogic Server*.

Simple Object Access Protocol (SOAP) 1.1 and 1.2

Simple Object Access Protocol (SOAP) is a lightweight XML-based protocol used to exchange information in a decentralized, distributed environment.

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

See the *Simple Object Access Protocol (SOAP)* specification, described at http://www.w3.org/TR/SOAP. 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.

SOAP Over JMS Transport 1.0

SOAP over JMS services transport is supported as a connection protocol for JAX-WS WebLogic web services.

For JAX-WS, this feature supports the new *W3C SOAP over Java Message Service 1.0* standard (February 2012), available at: http://www.w3.org/TR/soapjms/

For more information, see Using JMS Transport as the Connection Protocol in *Developing JAX-WS Web Services for Oracle WebLogic Server*.

SOAP with Attachments API for Java (SAAJ) 1.3

The *SOAP with Attachments API for Java (SAAJ)* describes how developers can produce and consume messages conforming to the SOAP 1.1 specification and SOAP with Attachments notes.

See the *SOAP with Attachments API for Java (SAAJ)* specification, described at https://jcp.org/en/jsr/detail?id=67.

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 Application Description Language (WADL) 2009 Membership Submission

Web Application Description Language (WADL) is an XML-based specification that provides a machine-readable description of HTTP-based Web applications. Developers of WebLogic web services do not need to create the WADL files; you generate these files automatically as part of the WebLogic web services development process.

Namespace: `http://wadl.dev.java.net/2009/02/wadl.xsd`

See *Web Application Description Language (WADL)* specification at `http://www.w3.org/Submission/wadl`.

See *Developing and Securing RESTful Web Services for Oracle WebLogic Server*.

Web Services Addressing (WS-Addressing) 1.0 and 2004/08 Member Submission

The *Web Services Addressing (WS-Addressing) Core* provides transport-neutral mechanisms to address web services and messages.

Namespaces:

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

`http://www.w3.org/2007/05/addressing/metadata`

See the *Web Services Addressing (WS-Addressing) Core* specification, described at `http://www.w3.org/TR/ws-addr-core`. 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.

In addition to 1.0, 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) Metadata* specification, described at `http://www.w3.org/TR/ws-addr-metadata`, defines how the abstract properties defined in *Web Services Addressing Core* are described using WSDL and how WS-Policy can be used to indicate the support of WS-Addressing by a web service.

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

The *Web Services Atomic Transaction (WS-AtomicTransaction)* 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 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.

See Using Web Services Atomic Transactions in *Developing JAX-WS Web Services for Oracle WebLogic Server*.

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

The *Web Services Coordination (WS-Coordination)* defines an extensible framework for providing protocols that coordinate the actions of distributed applications.

See 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. The WS-AtomicTransaction and WS-Coordination specifications define an extensible framework for coordinating distributed activities among a set of participants.

Web Services Description Language (WSDL) 1.1

Web Services Description Language (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.

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

See the *Web Services Description Language (WSDL)* specification at http://www.w3.org/TR/wsdl.

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.

See Developing WebLogic Web Services Starting from a WSDL File: Main Steps in *Developing JAX-WS Web Services for Oracle WebLogic Server*.

Web Services MakeConnection 1.1

The *Web Services MakeConnection* provides a mechanism for the transfer of messages between two endpoints when the sending endpoint is unable to initiate a new connection to the receiving endpoint. For example, to enable asynchronous web service invocation from behind a firewall.

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

See the *Web Services MakeConnection* specification at `http://docs.oasis-open.org/ws-rx/wsmc/200702/wsmc-1.1-spec-os.html`.

See Developing Asynchronous Clients in *Developing JAX-WS Web Services for Oracle WebLogic Server*.

Web Services Metadata for the Java Platform 2.1 (JSR-181)

Oracle recommends that you take advantage of the metadata annotations feature in Oracle WebLogic Server. To do so, you 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.

See http://docs.oracle.com/javase/8/docs/technotes/guides/language/annotations.html.

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 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.

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

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.

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`

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

See *Securing WebLogic Web Services for Oracle WebLogic Server*.

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

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.

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`

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`

See *Securing WebLogic Web Services for Oracle WebLogic Server*.

Web Services Reliable Messaging (WS-ReliableMessaging)

The *Web Services Reliable Messaging (WS-ReliableMessaging)* 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.

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

See the *Web Services Reliable Messaging (WS-ReliableMessaging)* specification at `http://docs.oasis-open.org/ws-rx/wsrm/200702`, 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.

See Using Web Services Reliable Messaging in *Developing JAX-WS Web Services for Oracle WebLogic Server*.

> **Note:**
>
> The WebLogic Server WS-ReliableMessaging supports backward compatibility with older versions of the specification. For example, a WS-ReliableMessaging 1.2 web service can be accessed by clients conforming to either the WS-ReliableMessaging 1.2 or 1.1 specifications. However, a WS-ReliableMessaging 1.2/1.1 client cannot communicate with a WS-ReliableMessaging 1.0 server. Note that WS-ReliableMessaging 1.2 (client or service) is supported on JAX-WS only.

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

The *Web Services Reliable Messaging Policy Assertion (WS-RM 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.

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

See:

- Version 1.2 (JAX-WS only): `http://docs.oasis-open.org/ws-rx/wsrmp/200702/wsrmp-1.2-spec-os.html`
- Version 1.1 (JAX-WS only): `http://docs.oasis-open.org/ws-rx/wsrmp/200702/wsrmp-1.1-spec-os-01.html`

See Using Web Services Reliable Messaging in *Developing JAX-WS Web Services for Oracle WebLogic Server*.

Web Services Secure Conversation Language (WS-SecureConversation)

The Web Services Secure Conversation Language (WS-SecureConversation) specification defines extensions that build on Web Services Security (WS-Security) 1.1 and 1.0 and Web Services Trust Language (WS-Trust) to provide secure communication across one or more messages.

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

Specifically, the specification defines mechanisms for establishing and sharing security contexts, and deriving keys from established security contexts (or any shared secret).

See:

- Version 1.4 (JAX-WS): `http://docs.oasis-open.org/ws-sx/ws-secureconversation/v1.4/ws-secureconversation.html`

See *Securing WebLogic Web Services for Oracle WebLogic Server*.

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).

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

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 1.0 and 1.1

See *Securing WebLogic Web Services for Oracle WebLogic Server*.

Web Services Security Policy (WS-SecurityPolicy) 1.3

The *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.

Namespace: `http://docs.oasis-open.org/ws-sx/ws-securitypolicy/200802`

See the *Web Services Security Policy (WS-SecurityPolicy)* specification at `http://docs.oasis-open.org/ws-sx/ws-securitypolicy/v1.3/ws-securitypolicy.html`.

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.

See *Securing WebLogic Web Services for Oracle WebLogic Server*.

Web Services Trust Language (WS-Trust)

The *Web Services Trust Language (WS-Trust)* defines extensions that provides a framework for requesting and issuing security tokens, and to broker trust relationships.

Version 1.4 Namespace: `http://docs.oasis-open.org/ws-sx/ws-trust/200802`

Version 1.3 Namespace: `http://docs.oasis-open.org/ws-sx/ws-trust/200512`

See the *Web Services Trust Language (WS-Trust)* specifications at:

- Version 1.4 (JAX-WS): https://www.oasis-open.org/standards#wstrustv1.4

See *Securing WebLogic Web Services for Oracle WebLogic Server*.

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

3

Using the Development and Administration Tools

Oracle provides helpful tools for developing and administering WebLogic web services for Oracle WebLogic Server, such as Oracle IDEs to develop web services, administration tools to manage, test, and monitor WebLogic Web services, Oracle WebLogic Scripting Tool, Java Management Extensions (JMX), and so on.

- Using Oracle IDEs to Develop Web Services
  Oracle JDeveloper and Oracle Enterprise Pack for Eclipse (OEPE) tools are available to develop web services.
- Using the Administration Tools to Manage, Test, and Monitor WebLogic Web Services
  Basic administration of web services is very similar to basic administration of standard Java Platform, Enterprise Edition (Jakarta EE) applications and modules. These standard tasks include deploying and monitoring the Enterprise application, configuring the policy files, and so on.
- Using Oracle Enterprise Manager Fusion Middleware Control
  The Oracle Enterprise Manager Fusion Middleware Control (Fusion Middleware Control) is a Web browser-based, graphical user interface that you can use to administer and monitor a farm.
- Using Oracle WebLogic Remote Console
  The WebLogic Remote Console is a web browser-based, graphical user interface that you use to manage a WebLogic Server domain, one or more WebLogic Server instances, clusters, and applications, including web services, that are deployed to the server or cluster.
- Using the Oracle WebLogic Scripting Tool
  The WebLogic Scripting Tool (WLST) is a command-line scripting interface that you can use to interact with and configure WebLogic Server domains and instances, as well as deploy Jakarta EE modules and applications (including web services) to a particular WebLogic Server instance. Using WLST, system administrators and operators can initiate, manage, and persist WebLogic Server configuration changes.
- Using Oracle WebLogic Server Ant Tasks
  WebLogic Server includes a variety of Ant tasks that you can use to centralize many of the configuration and administrative tasks into a single Ant build script. Use `wlserver`, `wlconfig`, and `wldeploy` for basic Ant tasks.
- Using the Java Management Extensions (JMX)
  A managed bean (MBean) is a Java bean that provides a Java Management Extensions (JMX) interface. JMX is the Jakarta EE solution for monitoring and managing resources on a network. Like SNMP and other management standards, JMX is a public specification and many vendors of commonly used monitoring products support it.
- Using the Jakarta EE Deployment API
  The Jakarta EE Deployment architecture defines the contracts that enable tools or application programmers to configure and deploy applications on any Jakarta EE platform product. The contracts define a uniform model between tools and Jakarta EE platform products for application deployment configuration and deployment.

- Using Web Services Apache Maven Goals
  Apache Maven is a software tool for building and managing Java-based projects. WebLogic Server provides support for Maven through the provisioning of plug-ins that enable you to perform various operations on WebLogic Server from within a Maven environment.

Using Oracle IDEs to Develop Web Services

Oracle JDeveloper and Oracle Enterprise Pack for Eclipse (OEPE) tools are available to develop 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. See Developing and Securing Web Services in *Developing Applications with 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.

Using the Administration Tools to Manage, Test, and Monitor WebLogic Web Services

Basic administration of web services is very similar to basic administration of standard Java Platform, Enterprise Edition (Jakarta EE) applications and modules. These standard tasks include deploying and monitoring the Enterprise application, configuring the policy files, and so on.

When you use the `jwsc` Ant task to compile and package a WebLogic web service, the task packages it as part of an Enterprise application. The web service itself is packaged inside the Enterprise application as a Web application WAR file, by default. However, if your JWS file implements a session bean then the web service is packaged as an EJB JAR file.

The standard tasks include:

- Deploying the Enterprise application that contains the web service.
- Starting and stopping the deployed Enterprise application.
- Configuring the Enterprise application and the archive file which implements the actual web service. You can configure general characteristics of the Enterprise application, such as the deployment order, or module-specific characteristics, such as session time-out for Web applications or transaction type for EJBs.
- Creating and updating the Enterprise application's deployment plan.
- Monitoring the Enterprise application.
- Testing the Enterprise application.

The following provides examples of administrative tasks are specific to web services:

- Configuring the policy files associated with a web service endpoint or its operations.
- Viewing the SOAP handlers associated with the web service.

- Viewing the WSDL of the web service.
- Creating a web service security configuration.

There are a variety of ways to administer Jakarta EE modules and applications that run on WebLogic Server, including web services, as described in the following sections.

Using Oracle Enterprise Manager Fusion Middleware Control

The Oracle Enterprise Manager Fusion Middleware Control (Fusion Middleware Control) is a Web browser-based, graphical user interface that you can use to administer and monitor a farm.

A *farm* is a collection of managed components. It can contain Oracle WebLogic Server domains, one or more Managed Servers and the Oracle Fusion Middleware system components that are installed, configured, and running in the domain.

Fusion Middleware Control organizes a wide variety of performance data and administrative functions into distinct, Web-based home pages for the farm, Oracle WebLogic Server domain, components, and applications. The Fusion Middleware Control home pages make it easy to locate the most important monitoring data and the most commonly used administrative functions—all from your Web browser.

For more information about managing, testing, and monitoring web services using the Enterprise Manager, see *Administering Web Services*.

Fusion Middleware Control is available as part of the Oracle Fusion Middleware product; it is not available to you if you purchase the standalone version of Oracle WebLogic Server. See Getting Started Using Oracle Enterprise Manager Fusion Middleware Control in *Administering Oracle Fusion Middleware*.

Using Oracle WebLogic Remote Console

The WebLogic Remote Console is a web browser-based, graphical user interface that you use to manage a WebLogic Server domain, one or more WebLogic Server instances, clusters, and applications, including web services, that are deployed to the server or cluster.

One instance of WebLogic Server in each domain is configured as an Administration Server. The Administration Server provides a central point for managing a WebLogic Server domain. All other WebLogic Server instances in a domain are called Managed Servers. In a domain with only a single WebLogic Server instance, that server functions both as Administration Server and Managed Server. The Administration Server hosts the WebLogic Remote Console, which is a Web Application accessible from any supported Web browser with network access to the Administration Server.

The following sections provide more details on the following topics:

- Invoking the Remote Console
- How Web Services Are Displayed In the Remote Console
- Creating a Web Services Security Configuration

Invoking the Remote Console

To invoke the WebLogic Remote Console in your browser, enter the following URL:

`http://hostname:port/rconsole`

`https://hostname:port/rconsole`

Where `hostname` and `port` match the values you set when you deployed Hosted WebLogic Remote Console.

For more information, see Oracle WebLogic Remote Console Online Help.

How Web Services Are Displayed In the Remote Console

Web services are typically deployed to WebLogic Server as part of an Enterprise Application. The Enterprise Application can be either archived as an EAR, or be in exploded directory format. The web service itself is almost always packaged as a Web Application; the only exception is if your JWS file implements a session bean in which case it is packaged as an EJB. The web service can be in archived format (WAR or EJB JAR file, respectively) or as an exploded directory.

It is not required that a web service be installed as part of an Enterprise application; it can be installed as just the Web Application or EJB. However, Oracle recommends that users install the web service as part of an Enterprise application. The WebLogic Ant task used to create a web service, `jwsc`, always packages the generated web service into an Enterprise application.

Creating a Web Services Security Configuration

When a deployed WebLogic web service has been configured to use message-level security (encryption and digital signatures, as described by the WS-Security specification), the web services runtime determines whether a web service security configuration is also associated with the service. This security configuration specifies information such as whether to use an X.509 certificate for identity, whether to use password digests, the keystore to be used for encryption, and so on. A single security configuration can be associated with many web services.

Using the Oracle WebLogic Scripting Tool

The WebLogic Scripting Tool (WLST) is a command-line scripting interface that you can use to interact with and configure WebLogic Server domains and instances, as well as deploy Jakarta EE modules and applications (including web services) to a particular WebLogic Server instance. Using WLST, system administrators and operators can initiate, manage, and persist WebLogic Server configuration changes.

See:

- Web Services Custom WLST Commands in *WLST Command Reference for Infrastructure Components*
- *Understanding the WebLogic Scripting Tool*

Using Oracle WebLogic Server Ant Tasks

WebLogic Server includes a variety of Ant tasks that you can use to centralize many of the configuration and administrative tasks into a single Ant build script. Use `wlserver`, `wlconfig`, and `wldeploy` for basic Ant tasks.

The Ant tasks can:

- Create, start, and configure a new WebLogic Server domain, using the `wlserver` and `wlconfig` Ant tasks.

- Deploy a compiled application to the newly-created domain, using the `wldeploy` Ant task.
- Generate web services and clients, and download a WSDL to a local directory.

The following table summarizes the steps to use the web services Ant tasks.

Table 3-1 Steps to Use the Web Services Ant Tasks

#	Step	Description
1	Set up your environment.	On Windows NT, execute the `setDomainEnv.cmd` command, located in your domain directory. The default location of WebLogic Server domains is *ORACLE_HOME*\user_projects\domains*domainName*, where *ORACLE_HOME* represents the directory you specified as the Oracle Home when you installed WebLogic Server and *domainName* is the name of your domain. On UNIX, execute the `setDomainEnv.sh` command, located in your domain directory. The default location of WebLogic Server domains is *ORACLE_HOME*/user_projects/domains/*domainName*, where *ORACLE_HOME* represents the directory you specified as the Oracle Home when you installed WebLogic Server and *domainName* is the name of your domain.

2	Create the `build.xml` file that contains a call to the web services Ant tasks.	The following example shows a simple `build.xml` file with a single target called `clean`:
		`<project name="my-webservice">`
		`  <target name="clean">`
		`  <delete>`
		`  <fileset dir="tmp" />`
		`  </delete>`
		`  </target>`
		`</project>`
		This `clean` target deletes all files in the `tmp` subdirectory. Later sections provide examples of specifying the Ant task in the `build.xml` file.

3	For each WebLogic web service Ant task you want to execute, add an appropriate task definition and target to the `build.xml` file using the `<taskdef>` and `<target>` elements.	The following example shows how to add the `jwsc` Ant task to the build file; the attributes of the task have been removed for clarity:
		`<taskdef name="jwsc"`
		`  classname="weblogic.wsee.tools.anttasks.JwscTask" />`
		`<target name="build-service">`
		`  <jwsc attributes go here...>`
		`  ...`
		`  </jwsc>`
		`</target>`
		Note: You can name the WebLogic web services Ant tasks anything you want by changing the value of the name attribute of the relevant `<taskdef>` element. For consistency, however, this document uses the names `jwsc`, `clientgen`, `wsdlc`, and `wsdlget` throughout.

4	Execute the Ant task or tasks specified in the `build.xml` file.	Type `ant` in the same directory as the `build.xml` file and specify the target. For example: `prompt> ant build-service`
5	Specify the context path and service URI used in the URL that invokes the web service. (Optional)	You can set this information in several ways, as described in Defining the Context Path of a WebLogic Web Service in *Developing JAX-WS Web Services for Oracle WebLogic Server*.

For more information, see:

- Ant Task Reference in *WebLogic Web Services Reference for Oracle WebLogic Server*
- The following sections in *Developing Applications for Oracle WebLogic Server*:
  - Using Ant Tasks to Configure and Use a WebLogic Server Domain
  - wldeploy Ant Task Reference
- Setting the Classpath for the WebLogic Ant Tasks
- Differences in Operating System Case Sensitivity When Manipulating WSDL and XML Schema Files

Setting the Classpath for the WebLogic Ant Tasks

Each WebLogic Ant task accepts a `classpath` attribute or element so that you can add new directories or JAR files to your current CLASSPATH environment variable.

The following example shows how to use the `classpath` attribute of the `jwsc` Ant task to add a new directory to the CLASSPATH variable:

```
<jwsc srcdir="MyJWSFile.java"
      classpath="${java.class.path};my_fab_directory"
   ...
</jwsc>
```

The following example shows how to add to the CLASSPATH by using the `<classpath>` element:

```
<jwsc ...>
   <classpath>
       <pathelement path="${java.class.path}" />
       <pathelement path="my_fab_directory" />
   </classpath>
...
</jwsc>
```

The following example shows how you can build your CLASSPATH variable outside of the WebLogic web service Ant task declarations, then specify the variable from within the task using the `<classpath>` element:

```
<path id="myClassID">
   <pathelement path="${java.class.path}"/>
   <pathelement path="${additional.path1}"/>
   <pathelement path="${additional.path2}"/>
</path>
<jwsc ....>
   <classpath refid="myClassID" />
...
</jwsc>
```

> **Note:**
>
> The Java Ant utility included in WebLogic Server uses the `ant` (UNIX) or `ant.bat` (Windows) configuration files in the *`WL_HOME`*`\server\bin` directory to set various Ant-specific variables, where *`WL_HOME`* is the top-level directory of your WebLogic Server installation If you need to update these Ant variables, make the relevant changes to the appropriate file for your operating system.

Differences in Operating System Case Sensitivity When Manipulating WSDL and XML Schema Files

Many WebLogic web service Ant tasks have attributes that you can use to specify a file, such as a WSDL or an XML Schema file.

The Ant tasks process these files in a case-sensitive way. This means that if, for example, the XML Schema file specifies two user-defined types whose names differ only in their capitalization (for example, `MyReturnType` and `MYRETURNTYPE`), the `clientgen` Ant task correctly generates two separate sets of Java source files for the Java representation of the user-defined data type: `MyReturnType.java` and `MYRETURNTYPE.java`.

However, compiling these source files into their respective class files might cause a problem if you are running the Ant task on Microsoft Windows, because Windows is a case *insensitive* operating system. This means that Windows considers the files `MyReturnType.java` and `MYRETURNTYPE.java` to have the same name. So when you compile the files on Windows, the second class file overwrites the first, and you end up with only one class file. The Ant tasks, however, expect that *two* classes were compiled, thus resulting in an error similar to the following:

```
c:\src\com\bea\order\MyReturnType.java:14:
class MYRETURNTYPE is public, should be declared in a file named   MYRETURNTYPE.java
public class MYRETURNTYPE
       ^
```

To work around this problem rewrite the XML Schema so that this type of naming conflict does not occur, or if that is not possible, run the Ant task on a case sensitive operating system, such as Unix.

Using the Java Management Extensions (JMX)

A managed bean (MBean) is a Java bean that provides a Java Management Extensions (JMX) interface. JMX is the Jakarta EE solution for monitoring and managing resources on a network. Like SNMP and other management standards, JMX is a public specification and many vendors of commonly used monitoring products support it.

WebLogic Server provides a set of MBeans that you can use to configure, monitor, and manage WebLogic Server resources through JMX. WebLogic web services also have their own set of MBeans that you can use to perform some web service administrative tasks.

There are two types of MBeans: runtime (for read-only monitoring information) and configuration (for configuring the web service after it has been deployed).

The configuration web services MBeans are:

- `WebserviceSecurityConfigurationMBean`
- `WebserviceCredentialProviderMBean`
- `WebserviceSecurityMBean`
- `WebserviceSecurityTokenMBean`
- `WebserviceTimestampMBean`
- `WebserviceTokenHandlerMBean`

The runtime web services MBeans are:

- `WseeRuntimeMBean`
- `WseeHandlerRuntimeMBean`
- `WseePortRuntimeMBean`
- `WseeOperationRuntimeMBean`
- `WseePolicyRuntimeMBean`

See *MBean Reference for Oracle WebLogic Server* and the following sections in *Developing Custom Management Utilities Using JMX for Oracle WebLogic Server*:

- Understanding WebLogic Server MBeans
- Accessing WebLogic Server MBeans with JMX
- Managing a Domain's Configuration with JMX

Using the Jakarta EE Deployment API

The Jakarta EE Deployment architecture defines the contracts that enable tools or application programmers to configure and deploy applications on any Jakarta EE platform product. The contracts define a uniform model between tools and Jakarta EE platform products for application deployment configuration and deployment.

The *J2EE Application Deployment* specification (JSR-88), described at `http://jcp.org/en/jsr/detail?id=88`, defines a standard API that you can use to configure an application for deployment to a target application server environment.

The Deployment architecture makes it easier to deploy applications: Deployers do not have to learn all the features of many different Jakarta EE deployment tools in order to deploy an application on many different Jakarta EE platform products.

See *Deploying Applications to Oracle WebLogic Server* for more information.

Using Web Services Apache Maven Goals

Apache Maven is a software tool for building and managing Java-based projects. WebLogic Server provides support for Maven through the provisioning of plug-ins that enable you to perform various operations on WebLogic Server from within a Maven environment.

WebLogic Server provides support for the following web services Maven goals.

Table 3-2 Web Services Maven Goals

Maven Goal	Description
`ws-clientgen`	Generates client web service artifacts from a WSDL.
`ws-wsdlc`	Generates a set of artifacts and a partial Java implementation of the web service from a WSDL.
`ws-jwsc`	Builds a JAX-WS web service.

See Using the WebLogic Development Maven Plug-in in *Developing Applications for Oracle WebLogic Server* for complete documentation.

4

Roadmap and Related Information

Understand how to implement WebLogic web services for Oracle WebLogic Server using a roadmap that lists common tasks for creating, deploying, and invoking WebLogic web services, along with a summary of related documentation.

- Roadmap for Implementing WebLogic Web Services
  The roadmap provides common tasks for creating, deploying, and invoking WebLogic web services, such as reviewing the supported standards, running the samples, developing and administering web services using JAX-WS, and more.
- 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.
- Related Documentation—WebLogic Server Application Development
  For comprehensive guidelines for developing, deploying, and monitoring WebLogic Server applications, refer to the documents such as *Developing Applications for Oracle WebLogic Server*, *Developing Web Applications, Servlets, and JSPs for Oracle WebLogic Server*, *Developing XML Applications for Oracle WebLogic Server*, and so on.

Roadmap for Implementing WebLogic Web Services

The roadmap provides common tasks for creating, deploying, and invoking WebLogic web services, such as reviewing the supported standards, running the samples, developing and administering web services using JAX-WS, and more.

Table 4-1 Roadmap for Implementing WebLogic Web Services

Task	More Information
Review supported standards	Features and Standards Supported by WebLogic Web Services
Run samples	Examples for Jakarta EE Web Service Developers
Develop and administer web services using JAX-WS	*Developing JAX-WS Web Services for Oracle WebLogic Server*
Develop and administer RESTful web services using JAX-RS	*Developing and Securing RESTful Web Services for Oracle WebLogic Server*
Secure the web service—Oracle Web Services Manager (OWSM) policies	*Securing Web Services and Managing Policies with Oracle Web Services Manager*
Secure the web service—WebLogic web service policies	*Securing WebLogic Web Services for Oracle WebLogic Server*
Attach OWSM policies	• Attaching Policies in *Securing Web Services and Managing Policies with Oracle Web Services Manager* • Attaching Policies in *Developing Applications with Oracle JDeveloper*
Attach WebLogic web service policies	• Using Oracle Web Service Manager Security Policies in *Securing WebLogic Web Services for Oracle WebLogic Server* • Attaching Policies in *Developing Applications with Oracle JDeveloper*
Administer web services—Fusion Middleware Control	*Administering Web Services*

Table 4-1 (Cont.) Roadmap for Implementing WebLogic Web Services

Task	More Information
Test web services	• Testing Web Services in *Administering Web Services* • Testing and Debugging Web Services in *Developing Applications with Oracle JDeveloper*
Monitor web service performance	• Monitoring and Auditing Web Services in *Administering Web Services*
Create custom OWSM policy file	Creating Custom Assertions in *Developing Extensible Applications with Oracle Web Services Manager*
Create custom WebLogic web service policy file	Creating and Using a Custom Policy File in *Securing WebLogic Web Services for Oracle WebLogic Server*
Interoperate WebLogic and Oracle WSM web service policies	*Interoperability Guide for Oracle Web Services Manager*
Upgrade	Upgrading WebLogic Web Services in *Upgrading Oracle WebLogic Server*

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.

Table 4-2 WebLogic Web Services Documentation Set

Document	Description
Understanding Web Services	Develop web services for Oracle Fusion Middleware 14*c*.
Understanding WebLogic Web Services for Oracle WebLogic Server (This Document)	Introduces WebLogic web services, the standards that are supported, interoperability information, and relevant samples and documentation.
Understanding Oracle Web Services Manager	Introduces WebLogic web services, the standards that are supported, interoperability information, and relevant samples and documentation.
Developing JAX-WS Web Services for Oracle WebLogic Server	Describes how to develop WebLogic web services 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.
Developing and Securing RESTful Web Services for Oracle WebLogic Server	Describes how to develop WebLogic web services that conform to the Representational State Transfer (REST) architectural style using Java API for RESTful Web Services (JAX-RS).
Securing WebLogic Web Services for Oracle WebLogic Server	Describes how to develop and configure message-level (digital signatures and encryption), transport-level, and access control security for a web service.
Securing Web Services and Managing Policies with Oracle Web Services Manager	Describes how to secure web services using Oracle Web Services Manager (OWSM) policies.
Administering Web Services	Administer web services for Oracle Fusion Middleware 14*c*.
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.
Interoperability Guide for Oracle Web Services Manager	Interoperate with OWSM.

Table 4-2 (Cont.) WebLogic Web Services Documentation Set

Document	Description
Developing Extensible Applications for Oracle Web Services Manager	Develop custom assertions for OWSM.

Related Documentation—WebLogic Server Application Development

For comprehensive guidelines for developing, deploying, and monitoring WebLogic Server applications, refer to the documents such as *Developing Applications for Oracle WebLogic Server*, *Developing Web Applications, Servlets, and JSPs for Oracle WebLogic Server*, *Developing XML Applications for Oracle WebLogic Server*, and so on.

Table 4-3 Related Documentation—WebLogic Server Application Development

Review this document . . .	To learn how to . . .
Developing Applications for Oracle 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.
Developing Enterprise JavaBeans, Version 2.1, for Oracle WebLogic Server	Develop EJBs that are deployed and run on WebLogic Server.
Developing XML Applications 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.
Tuning Performance of Oracle WebLogic Server	Monitor and improve the performance of WebLogic Server applications.
System Administration in *Understanding Oracle WebLogic Server*	Administer WebLogic Server and its deployed applications.

5

Interoperability with Microsoft WCF/.NET

Oracle performs interoperability testing, in conjunction with Microsoft, to ensure that WebLogic web services for Oracle WebLogic Server can access and consume web services created using Microsoft Windows Communication Foundation (WCF)/.NET 3.0, 3.5, and Framework 4.0, and vice versa.
Table 5-1 describes the interoperability tests that were completed on JAX-WS web services.

Table 5-1 Completed Interoperability Tests

Area	Interoperability Guidelines
Basic and complex data types	Basic Data Types Interoperability Guidelines
WS-I Basic Profile 2.0, 1.2, and 1.1	Basic Profile Interoperability Guidelines **Note**: WS-I Basic Profile 2.0 and 1.2 applies to JAX-WS only. WS-I Basic Profile 1.1 applies to JAX-WS web services.
Web Services Reliable Secure Profile (WS-RSP) 1.0	Web Services Reliable Secure Profile Interoperability Guidelines
Web Services Security (WS-Security) 1.0 and 1.1	WS-Security Interoperability Guidelines
Web Services Security Policy (WS-SecurityPolicy) 1.2	WS-SecurityPolicy Interoperability Guidelines
Web Services Secure Conversation Language (WS-SecureConversation) 1.3	WS-SecureConversation Interoperability Guidelines
Web Services Policy Framework (WS-Policy) 1.5	No interoperability restrictions.
Web Services Addressing (WS-Addressing) 0.9 and 1.0	N/A
Message Transmission Optimization Mechanism (MTOM)	N/A
SAML Assertions	Using SAML Assertions Referenced from SignedInfo

In addition, the following combined features were tested:

- MTOM and WS-Security
- WS-ReliableMessaging and MTOM
- WS-ReliableMessaging 1.2 and WS-Addressing 1.0 (JAX-WS)
- WS-ReliableMessaging 1.1 and WS-Addressing 1.0 (JAX-WS)
- WS-ReliableMessaging 1.2 and WS-SecureConversation 1.4
- WS-ReliableMessaging 1.1 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`.
- Basic Profile Interoperability Guidelines
  Follow the basic profile interoperability guidelines to test the WS-I Basic Profiles.
- Web Services Reliable Secure Profile Interoperability Guidelines
  The Web Services Reliable Secure Profile implementations for WebLogic web services and Microsoft .NET Web are compatible with few caveats.
- WS-Security Interoperability Guidelines
  WebLogic Server lists interoperability guidelines for WS-Security, such as defining the security policies, Microsoft .NET 3.0/3.5 guidelines, and so on.
- WS-SecurityPolicy Interoperability Guidelines
  WebLogic Server provides WS-SecurityPolicy interoperability guidelines to be followed.
- WS-SecureConversation Interoperability Guidelines
  Use the interoperability guidelines for WS-SecureConversation, such as usage of `<sp:SignBeforeEncrypt>`, `setCompatibilityPreference("msft")` method, and so on.
- 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.

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`.

Basic Profile Interoperability Guidelines

Follow the basic profile interoperability guidelines to test the WS-I Basic Profiles.

The WS-I Basic Profile 1.2 and 2.0 profiles were tested between WebLogic web services JAX-WS and the Microsoft .NET Framework 4.0. No interoperability restrictions were found.

Web Services Reliable Secure Profile Interoperability Guidelines

The Web Services Reliable Secure Profile implementations for WebLogic web services and Microsoft .NET Web are compatible with few caveats.

- 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 only supported for WebLogic web service JAX-WS clients and Microsoft .NET services.

WS-Security Interoperability Guidelines

WebLogic Server lists interoperability guidelines for WS-Security, such as defining the security policies, Microsoft .NET 3.0/3.5 guidelines, and so on.

- 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.

WS-SecurityPolicy Interoperability Guidelines

WebLogic Server provides WS-SecurityPolicy interoperability guidelines to be followed.

In this release, WebLogic Server and Microsoft .NET 3.5 support Web Services Security Policy (WS-SecurityPolicy) 1.3. 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.

WS-SecureConversation Interoperability Guidelines

Use the interoperability guidelines for WS-SecureConversation, such as usage of `<sp:SignBeforeEncrypt>`, `setCompatibilityPreference("msft")` method, and so on.

- 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 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");
    ```

  For examples, see Example 5-1 and #unique_76/unique_76_Connect_42_BABFFEIF.

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.

See https://groups.oasis-open.org/higherlogic/ws/public/document?document_id=16768.

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 5-1 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);
      }
}
```

6

Examples for Jakarta EE Web Service Developers

Oracle provides a variety of samples that web service developers can use to learn more about WebLogic web services for Oracle WebLogic Server.

- Samples for WebLogic Web Service Developers
  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.
- Additional Web Services Samples Available for Download
  The additional Web services samples include Oracle-certified ones, and the samples submitted by fellow developers. Your use rights and restrictions for each sample code item described in the applicable license agreement.

Samples for WebLogic Web Service Developers

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.

Web services samples include:

- Web Services Samples in the WebLogic Server Distribution
- Avitek Medical Records Application (MedRec) and Tutorials

Web Services Samples in the WebLogic Server Distribution

WebLogic Server optionally installs API code examples in the *ORACLE_HOME*\wlserver\samples\server\examples\src\examples\webservices directory, where *ORACLE_HOME* represents the directory in which you installed WebLogic Server. See Sample Applications and Code Examples in *Understanding Oracle WebLogic Server*.

Avitek Medical Records Application (MedRec) and Tutorials

MedRec is an end-to-end sample Jakarta 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 Jakarta EE features, and highlights Oracle-recommended best practices. MedRec is optionally installed with the WebLogic Server installation. You can start MedRec from the *ORACLE_HOME*\user_projects\domains\medrec directory, where *ORACLE_HOME* is the directory you specified as the Oracle Home when you installed Oracle WebLogic Server. See Sample Applications and Code Examplesin *Understanding Oracle WebLogic Server*.

Additional Web Services Samples Available for Download

The additional Web services samples include Oracle-certified ones, and the samples submitted by fellow developers. Your use rights and restrictions for each sample code item described in the applicable license agreement.

Additional API samples for download can be found at http://www.oracle.com/technetwork/indexes/samplecode/index.html.