## Contents

### Preface

- Documentation Accessibility: xiii
- Conventions: xiii

### What's New in This Guide

- New and Changed Features for 12c (12.2.1.x): xiv
- New and Changed Features for 12c (12.2.1): xiv
- New and Changed Features for 12c (12.1.3): xiv
- New and Changed Features for 12c (12.1.2): xiv

### 1 Introduction

### 2 Ant Task Reference

- Overview of WebLogic Web Services Ant Tasks: 2-1
- clientgen: 2-2
  - Taskdef Classname: 2-3
  - Child Elements
    - binding: 2-3
    - jmstransportclient: 2-4
    - xmlcatalog: 2-5
  - Attributes: 2-5
  - Examples: 2-12
- jwsc: 2-14
  - Taskdef Classname: 2-16
  - Child Elements
    - binding: 2-16
    - clientgen: 2-18
    - descriptor: 2-22
    - jmstransportservice: 2-23
    - jws: 2-24
3 JWS Annotation Reference

Overview of JWS Annotation Tags 3-1
Web Services Metadata Annotations (JSR-181) 3-3
JAX-WS Annotations (JSR-224) 3-4
JAXB Annotations (JSR-222) 3-5
Common Annotations (JSR-250) 3-6
WebLogic-Specific Annotations 3-6
  com.oracle.webservices.api.jms.JMSTransportClient 3-9
  com.oracle.webservices.api.jms.JMSTransportService 3-11
  weblogic.jws.AsyncFailure 3-12
  weblogic.jws.AsyncResponse 3-14
  weblogic.jws.Binding 3-16
  weblogic.jws.BufferQueue 3-17
    Description 3-17
    Attributes 3-17
    Example 3-17
weblogic.jws.Callback
  Description 3-18
  Example 3-18

weblogic.jws.CallbackMethod
  Description 3-19
  Attributes 3-20
  Example 3-20

weblogic.jws.CallbackService
  Description 3-20
  Attributes 3-21
  Example 3-21

weblogic.jws.Context
  Description 3-22
  Example 3-22

weblogic.jws.Conversation
  Description 3-22
  Attributes 3-23
  Example 3-23

weblogic.jws.Conversational
  Description 3-24
  Attributes 3-25
  Example 3-26

weblogic.jws.FileStore
  Description 3-27
  Attributes 3-27

weblogic.jws.MessageBuffer
  Description 3-27
  Attributes 3-28
  Example 3-29

weblogic.jws.Policies
  Description 3-29
  Example 3-29

weblogic.jws.Policy
  Description 3-30
  Attributes 3-31
  Example 3-31

weblogic.jws.ReliabilityBuffer
  Description 3-31
  Attributes 3-32
  Example 3-33

weblogic.jws.ReliabilityErrorHandler
<table>
<thead>
<tr>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Attributes</td>
<td>3-47</td>
</tr>
<tr>
<td>Example</td>
<td>3-48</td>
</tr>
<tr>
<td><code>we3blogic.jws.security.RolesAllowed</code></td>
<td>3-48</td>
</tr>
<tr>
<td>Description</td>
<td>3-48</td>
</tr>
<tr>
<td>Attributes</td>
<td>3-48</td>
</tr>
<tr>
<td>Example</td>
<td>3-48</td>
</tr>
<tr>
<td><code>weblogic.jws.security.RolesReferenced</code></td>
<td>3-49</td>
</tr>
<tr>
<td>Description</td>
<td>3-49</td>
</tr>
<tr>
<td>Example</td>
<td>3-49</td>
</tr>
<tr>
<td><code>weblogic.jws.security.RunAs</code></td>
<td>3-49</td>
</tr>
<tr>
<td>Description</td>
<td>3-50</td>
</tr>
<tr>
<td>Attributes</td>
<td>3-50</td>
</tr>
<tr>
<td>Example</td>
<td>3-50</td>
</tr>
<tr>
<td><code>weblogic.jws.security.SecurityRole</code></td>
<td>3-50</td>
</tr>
<tr>
<td>Description</td>
<td>3-50</td>
</tr>
<tr>
<td>Attributes</td>
<td>3-51</td>
</tr>
<tr>
<td>Example</td>
<td>3-51</td>
</tr>
<tr>
<td><code>weblogic.jws.security.SecurityRoleRef</code></td>
<td>3-52</td>
</tr>
<tr>
<td>Description</td>
<td>3-52</td>
</tr>
<tr>
<td>Attributes</td>
<td>3-52</td>
</tr>
<tr>
<td>Example</td>
<td>3-52</td>
</tr>
<tr>
<td><code>weblogic.jws.security.UserDataConstraint</code></td>
<td>3-53</td>
</tr>
<tr>
<td>Description</td>
<td>3-53</td>
</tr>
<tr>
<td>Attributes</td>
<td>3-53</td>
</tr>
<tr>
<td>Example</td>
<td>3-53</td>
</tr>
<tr>
<td><code>weblogic.jws.security.WssConfiguration</code></td>
<td>3-54</td>
</tr>
<tr>
<td>Description</td>
<td>3-54</td>
</tr>
<tr>
<td>Attributes</td>
<td>3-55</td>
</tr>
<tr>
<td>Example</td>
<td>3-55</td>
</tr>
<tr>
<td><code>weblogic.jws.soap.SOAPBinding</code></td>
<td>3-55</td>
</tr>
<tr>
<td>Description</td>
<td>3-55</td>
</tr>
<tr>
<td>Attributes</td>
<td>3-56</td>
</tr>
<tr>
<td>Example</td>
<td>3-56</td>
</tr>
<tr>
<td><code>weblogic.jws.security.SecurityRoles</code> (deprecated)</td>
<td>3-57</td>
</tr>
<tr>
<td>Description</td>
<td>3-57</td>
</tr>
<tr>
<td>Attributes</td>
<td>3-58</td>
</tr>
<tr>
<td>Example</td>
<td>3-58</td>
</tr>
<tr>
<td><code>weblogic.jws.security.SecurityIdentity</code> (deprecated)</td>
<td>3-59</td>
</tr>
<tr>
<td>Description</td>
<td>3-59</td>
</tr>
<tr>
<td>Attributes</td>
<td>3-59</td>
</tr>
</tbody>
</table>
4  Web Service Reliable Messaging Policy Assertion Reference

Overview of a WS-Policy File That Contains Web Service Reliable Messaging Assertions 4-1

WS-Policy File With Web Service Reliable Messaging Assertions—Version 1.2 and 1.1 4-2

Example of a WS-Policy File With Web Service Reliable Messaging Assertions 4-2

Element Descriptions 4-2

wsp:Policy 4-3
wsrmp:DeliveryAssurance 4-3
wsrmp:RMAssertion 4-3
wsrmp:SequenceSTR 4-3
wsrmp:SequenceTransportSecurity 4-4

WS-Policy File With Web Service Reliable Messaging Assertions—Version 1.0 (Deprecated) 4-4

Example of a WS-Policy File With Web Service Reliable Messaging Assertions 4-4

Element Description 4-4

beapolicy:Expires 4-5
Using MessageParts To Specify Parts of the SOAP Messages that Must Be Encrypted or Signed

XPath 1.0

Pre-Defined wsp:Body() Function

WebLogic-Specific Header Functions

7 WebLogic Web Service Deployment Descriptor Schema Reference

Overview of weblogic-webservices.xml

Example of a weblogic-webservices.xml Deployment Descriptor File

Element Descriptions

acknowledgement-interval

activation-config

auth-constraint

base-retransmission-interval

binding-version

buffer-retry-count

buffer-retry-delay

buffering-config

callback-protocol

connection-factory-jndi-name

customized

default-logical-store-name

delivery-mode

deployment-listener-list

deployment-listener

destination-name

destination-type

enable-http-wsdl-access

enabled

exposed

fastinfoset

flowType

http-flush-response

http-response-buffersize

inactivity-timeout

jndi-connection-factory-name

jndi-context-parameter

jndi-initial-context-factory

jndi-url

<table>
<thead>
<tr>
<th>Term</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>webservice-serviceuri</td>
<td>7-19</td>
</tr>
<tr>
<td>webservice-type</td>
<td>7-19</td>
</tr>
<tr>
<td>wsat-config</td>
<td>7-19</td>
</tr>
<tr>
<td>wsdl</td>
<td>7-20</td>
</tr>
<tr>
<td>wsdl-publish-file</td>
<td>7-20</td>
</tr>
</tbody>
</table>
Preface

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

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.

Accessible Access to Oracle Support

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

Conventions

The following text conventions are used in this document:

<table>
<thead>
<tr>
<th>Convention</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>boldface</strong></td>
<td>Boldface type indicates graphical user interface elements associated with an action, or terms defined in text or the glossary.</td>
</tr>
<tr>
<td><em>italic</em></td>
<td>Italic type indicates book titles, emphasis, or placeholder variables for which you supply particular values.</td>
</tr>
<tr>
<td><strong>monospace</strong></td>
<td>Monospace type indicates commands within a paragraph, URLs, code in examples, text that appears on the screen, or text that you enter.</td>
</tr>
</tbody>
</table>
What's New in This Guide

The following topics introduce the new and changed features of WebLogic web services in Oracle Fusion Middleware 12c (12.2.1), and provides pointers to additional information.

New and Changed Features for 12c (12.2.1.x)

For Oracle Fusion Middleware 12c (12.2.1.x), this document does not contain any new or changed features. For a comprehensive listing of the new WebLogic Server features introduced in this release, see What's New in Oracle WebLogic Server.

New and Changed Features for 12c (12.2.1)

The JAX-RPC API is deprecated in WebLogic Server as of release 12.2.1.

New and Changed Features for 12c (12.1.3)

For Oracle Fusion Middleware 12c (12.1.3), this document does not contain any new or changed features.

New and Changed Features for 12c (12.1.2)

Oracle Fusion Middleware 12c (12.1.2) includes the following new and changed features for this document.

- Ant task feature support for SOAP over JMS transport, including the following new child elements:
  - `<jmstransportclient>` child element of `clientgen` to configure SOAP over JMS transport on the client, as described in `jmstransportclient`.
  - `<jmstransportservice>` child element of `jwsc` to configure SOAP over JMS transport on the service, as described in `jmstransportservice`.

- Annotation support for SOAP over JMS transport, including the following:
  - `com.oracle.webservices.api.jms.JMSTransportClient` annotation, as described in `com.oracle.webservices.api.jms.JMSTransportClient`.
  - `com.oracle.webservices.api.jms.JMSTransportService` annotation, as described in `com.oracle.webservices.api.jms.JMSTransportService`.

- Annotation support for attaching Oracle Web Services Manager (OWSM) security policies to WebLogic web service clients, as well as WebLogic web services. Annotation support includes:
- `weblogic.wsee.jws.jaxws.owsm.Property` annotation to override configuration properties when attaching an OWSM policy, as described in `weblogic.wsee.jws.jaxws.owsm.Property`.


- New deployment descriptor elements to support SOAP over JMS transport, as described in WebLogic Web Service Deployment Descriptor Schema Reference.
Introduction

This chapter lists the reference information that is available to software developers who develop WebLogic web services.

The following table summarizes the topics described in this document.

**Table 1-1  WebLogic Web Service Reference Topics**

<table>
<thead>
<tr>
<th>This Reference Topic . . .</th>
<th>Describes . . .</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ant Task Reference</td>
<td>WebLogic web services Ant tasks.</td>
</tr>
<tr>
<td>JWS Annotation Reference</td>
<td>JWS annotations that you can use in the JWS file that implements your web service.</td>
</tr>
<tr>
<td>Web Service Reliable</td>
<td>Policy assertions you can add to a WS-Policy file to configure the web service reliable messaging feature of a WebLogic web service.</td>
</tr>
<tr>
<td>Messaging Policy Assertion Reference</td>
<td>Policy assertions you can add to a WS-Policy file to configure the web service MakeConnection feature of a WebLogic web service.</td>
</tr>
</tbody>
</table>
| Oracle Web Services Security Policy Assertion Reference | Policy assertions you can add to a WS-Policy file to configure the message-level (digital signatures and encryption) security of a WebLogic web service, using a proprietary Oracle security policy schema.  
**Note:** You may prefer to use files that conform to the OASIS WS-SecurityPolicy specification, as described in Configuring Message-Level Security in Securing WebLogic Web Services for Oracle WebLogic Server. |
| WebLogic Web Service      | Elements in the WebLogic-specific web services deployment descriptor weblogic-webservices.xml. |
| Deployment Descriptor     | Reference       |

For an overview of WebLogic web services, samples, and related documentation, see *Understanding WebLogic Web Services for Oracle WebLogic Server*. 

---

1-1
WebLogic web services 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.

- Overview of WebLogic Web Services Ant Tasks
- clientgen
- jwsc
- wsdlc
- wsdiget

Overview of WebLogic Web Services Ant Tasks

Ant is a Java-based build tool, similar to the make command but much more powerful. Ant uses XML-based configuration files (called build.xml by default) to execute tasks written in Java. Oracle provides a number of Ant tasks that help you generate important web service-related artifacts. The Apache Web site provides other useful Ant tasks for packaging EAR, WAR, and EJB JAR files. See the Apache Ant Manual at http://jakarta.apache.org/ant/manual/.

Note:

The Apache Jakarta Web site publishes online documentation for only the most current version of Ant, which might be different from the version of Ant that is bundled with WebLogic Server. To determine the version of Ant that is bundled with WebLogic Server, run the following command after setting your WebLogic environment:

prompt> ant -version

To view the documentation for a specific version of Ant, download the Ant zip file from http://archive.apache.org/dist/ant/binaries/ and extract the documentation.

The following table provides an overview of the web service Ant tasks provided by Oracle.

Table 2-1 WebLogic Web Service Ant Tasks

<table>
<thead>
<tr>
<th>Ant Task</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>clientgen</td>
<td>Generates the Service stubs and other client-side artifacts used to invoke a web service.</td>
</tr>
</tbody>
</table>
### Table 2-1  (Cont.) WebLogic Web Service Ant Tasks

<table>
<thead>
<tr>
<th>Ant Task</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>jwsc</td>
<td>Compiles a Java web service (JWS)-annotated file into a web service.</td>
</tr>
<tr>
<td>wsdlc</td>
<td>Generates a partial web service implementation based on a WSDL file.</td>
</tr>
<tr>
<td>wsdlget</td>
<td>Downloads to the local directory a WSDL and its imported XML targets, such as XSD and WSDL files.</td>
</tr>
</tbody>
</table>

For detailed information about how to integrate and use these Ant tasks in your development environment to program a web service and a client application that invokes the web service, see:

- Using Oracle WebLogic Server Ant Tasks in *Understanding WebLogic Web Services for Oracle WebLogic Server*
- *Developing JAX-WS Web Services for Oracle WebLogic Server*
- *Oracle Fusion Middleware Developing JAX-RPC Web Services for Oracle WebLogic Server*

---

**clientgen**

The `clientgen` Ant task generates, from an existing WSDL file, the client component files that client applications use to invoke both WebLogic and non-WebLogic web services.

The generated artifacts for **JAX-WS** web services include:

- The Java class for the `Service` interface implementation for the particular web service you want to invoke.
- JAXB data binding artifacts.
- The Java class for any user-defined XML Schema data types included in the WSDL file.

The generated artifacts for **JAX-RPC** web services include:

- The Java class for the `Stub` and `Service` interface implementations for the particular web service you want to invoke.
- The Java source code for any user-defined XML Schema data types included in the WSDL file.
- The JAX-RPC mapping deployment descriptor file which contains information about the mapping between the Java user-defined data types and their corresponding XML Schema types in the WSDL file.
- A client-side copy of the WSDL file.

Two types of client applications use the generated artifacts of `clientgen` to invoke web services:

- Stand-alone Java clients that do not use the Java Platform, Enterprise Edition (Java EE) Version 5 client container.
- Java EE clients, such as EJBs, JSPs, and web services, that use the Java EE client container.
By default, the clientgen Ant task generates client artifacts for a JAX-RPC web service. If you are generating client artifacts for a JAX-WS web service, you can set the type attribute to JAXWS. For example: `type="JAXWS"`.

You typically use the `destDir` attribute of clientgen to specify the directory into which all the artifacts should be generated, and then compile the generate Java files yourself using the `javac` Ant task. However, clientgen also provides a `destFile` attribute if you want the Ant task to compile the Java files for you and package them, along with the other generated artifacts, into the specified JAR file. You must specify one of either `destFile` or `destDir`, although you cannot specify both.

The following sections provide more information about the clientgen Ant task:

- **Taskdef Classname**
- **Child Elements**
- **Attributes**
- **Examples**

## Taskdef Classname

The following shows the task definition for the clientgen classname which must appear in your Ant build file.

```xml
<taskdef name="clientgen" class="weblogic.wsee.tools.anttasks.ClientGenTask"/>
```

## Child Elements

The following sections describe the WebLogic-specific child elements for the clientgen Ant task.

- **binding**
- **jmstransportclient**
- **xmlcatalog**

### binding

Use the `<binding>` child element to specify one of the following:

- For JAX-WS, one or more customization files that specify one or more of the following:
- For JAX-RPC, one or more XMLBeans configuration files, which by convention end in `.xsdconfig`. Use this element if your web service uses Apache XMLBeans at [http://xmlbeans.apache.org/](http://xmlbeans.apache.org/) data types as parameters or return values.
You use the `<binding>` element the same way as the standard Ant FileSet data type, using the same attributes. For example, the following `<binding>` element specifies the JAX-WS custom binding declarations defined in the file `jaxws-binding.xml`:

```xml
<binding file="./jaxws-binding.xml"/>
```

The following example specifies the JAX-WS customization files that are located in the `${basedir}` directory:

```xml
<binding dir="${basedir}"/>
```

For information about the full set of attributes you can specify using the FileSet data type, obtain the documentation for the version of Ant you are using at [http://ant.apache.org/index.html](http://ant.apache.org/index.html) and navigate to the description of the FileSet type.

jmstransportclient

**Note:**

The `<jmstransportclient>` child element applies to JAX-WS only; this child element is not valid for JAX-RPC.

The `<jmstransportclient>` element enables and configures SOAP over JMS transport.

Optionally, you can configure the destination name, destination type, delivery mode, request and response queues, and other JMS transport properties, using the `<jmstransportclient>` element. For a complete list of JMS transport properties supported, see Configuring JMS Transport Properties in *Developing JAX-WS Web Services for Oracle WebLogic Server*.

The following example shows how to enable and configure JMS transport when generating the web service client using `clientgen`.

```xml
<target name="clientgen">
<clientgen
    wsdl="./WarehouseService.wsdl"
    destDir="clientclasses"
    packageName="client.warehouse"
    type="JAXWS">
    <jmstransportclient
        targetService="JWSCEndpointService"
        destinationName="com.oracle.webservices.jms.SoapJmsRequestQueue"
        jndiInitialContextFactory="weblogic.jndi.WLInitialContextFactory"
        jndiConnectionFactoryName="weblogic.jms.ConnectionFactory"
        jndiURL="t3://localhost:7001"
        deliveryMode="NON_PERSISTENT"
        timeToLive="60000"
        priority="1"
        messageType="TEXT"
        replyToName="com.oracle.webservices.jms.SoapJmsResponseQueue"
    />
</clientgen>
```
The `<xmlcatalog>` child element applies to JAX-WS only; this child element is not valid for JAX-RPC.

The `<xmlcatalog>` child element specifies the ID of an embedded XML catalog. The following shows the element syntax:

```xml
text_entity
</xmlcatalog>
```

The ID referenced by `<xmlcatalog>` must match the ID of an embedded XML catalog. You embed an XML catalog in the `build.xml` file using the following syntax:

```xml
text_entity
</xmlcatalog>
```

In the above syntax, `public_id` specifies the public identifier of the original XML resource (WSDL or XSD) and `uri` specifies the replacement XML resource.

The following example shows how to embed an XML catalog and reference it using `clientgen`. Relevant code lines are shown in **bold**.

```xml
<target name="clientgen">
  <clientgen
      type="JAXWS"
      wsdl="$wsdl1"
      destDir="${clientclasses.dir}"
      packageName="xmlcatalog.jaxws.clientgen.client"
      catalog="wsdlicatalog.xml">
    <xmlcatalog refid="wsimportcatalog"/>
  </clientgen>
</target>
<xmlcatalog id="wsimportcatalog">
  <entity publicid="http://helloservice.org/types/HelloTypes.xsd" location="${basedir}/HelloTypes.xsd"/>
</xmlcatalog>
```

See Using XML Catalogs in Developing JAX-WS Web Services for Oracle WebLogic Server.

### Attributes

The following table describes the WebLogic-specific attributes of the `clientgen` Ant task, and specifies whether they are valid for JAX-WS or JAX-RPC web services or both.
Table 2-2  WebLogic-specific Attributes of the clientgen Ant Task

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
<th>Data Type</th>
<th>Required?</th>
<th>JAX-WS, JAX-RPC, or Both?</th>
</tr>
</thead>
<tbody>
<tr>
<td>autoDetectWrapped</td>
<td>Specifies whether the clientgen Ant task should try to determine whether the parameters and return type of document-literal web services are of type wrapped or bare. When the clientgen Ant task parses a WSDL file to create the client stubs, it attempts to determine whether a document-literal web service uses wrapped or bare parameters and return types based on the names of the XML Schema elements, the name of the operations and parameters, and so on. Depending on how the names of these components match up, the clientgen Ant task makes a best guess as to whether the parameters are wrapped or bare. In some cases, however, you might want the Ant task to always assume that the parameters are of type bare; in this case, set the autoDetectWrapped attribute to False. Valid values for this attribute are True or False. The default value is True.</td>
<td>Boolean</td>
<td>No</td>
<td>JAX-RPC</td>
</tr>
<tr>
<td>catalog</td>
<td>Specifies an external XML catalog file. See Using XML Catalogs in Developing JAX-WS Web Services for Oracle WebLogic Server.</td>
<td>String</td>
<td>No</td>
<td>JAX-WS</td>
</tr>
<tr>
<td>copyWsdl</td>
<td>Controls whether the WSDL should be copied in the destination directory defined by destDir.</td>
<td>Boolean</td>
<td>No</td>
<td>JAX-WS</td>
</tr>
<tr>
<td>destDir</td>
<td>Directory into which the clientgen Ant task generates the client source code, WSDL, and client deployment descriptor files. You can set this attribute to any directory you want. However, if you are generating the client component files to invoke a web service from an EJB, JSP, or other web service, you typically set this attribute to the directory of the Java EE component which holds shared classes, such as META-INF for EJBs, WEB-INF/classes for Web Applications, or APP-INF/classes for Enterprise Applications. If you are invoking the web service from a stand-alone client, then you can generate the client component files into the same source code directory hierarchy as your client application code.</td>
<td>String</td>
<td>You must specify either the destFile or destDir attribute, but not both.</td>
<td>Both</td>
</tr>
<tr>
<td>Attribute</td>
<td>Description</td>
<td>Data Type</td>
<td>Required?</td>
<td>JAX-WS, JAX-RPC, or Both?</td>
</tr>
<tr>
<td>------------</td>
<td>-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
<td>-----------</td>
<td>-----------</td>
<td>---------------------------</td>
</tr>
<tr>
<td>destFile</td>
<td>Name of a JAR file or exploded directory into which the clientgen task packages the client source code, compiled classes, WSDL, and client deployment descriptor files. If you specify this attribute, the clientgen Ant task also compiles all Java code into classes. To create or update a JAR file, use a .jar suffix when specifying the JAR file, such as myclientjar.jar. If the attribute value does not have a .jar suffix, then the clientgen task assumes you are referring to a directory name. If you specify a JAR file or directory that does not exist, the clientgen task creates a new JAR file or directory.</td>
<td>String</td>
<td>You must specify either the destFile or destDir attribute, but not both.</td>
<td>Both</td>
</tr>
<tr>
<td>failonerror</td>
<td>Specifies whether the clientgen Ant task continues executing in the event of an error. Valid values for this attribute are True or False. The default value is True, which means clientgen continues executing even after it encounters an error.</td>
<td>Boolean</td>
<td>No</td>
<td>Both</td>
</tr>
</tbody>
</table>
Table 2-2  (Cont.) WebLogic-specific Attributes of the clientgen Ant Task

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
<th>Data Type</th>
<th>Required?</th>
<th>JAX-WS, JAX-RPC, or Both?</th>
</tr>
</thead>
<tbody>
<tr>
<td>generateAsyncMethods</td>
<td>Specifies whether the clientgen Ant task should include methods in the generated stubs that client applications can use to invoke a web service operation asynchronously. For example, if you specify True (which is also the default value), and one of the web service operations in the WSDL is called getQuote, then the clientgen Ant task also generates a method called getQuoteAsync in the stubs which client applications invoke instead of the original getQuote method. This asynchronous flavor of the operation also has an additional parameter, of data type weblogic.wsee.async.AsyncPreCallContext, that client applications can use to set asynchronous properties, contextual variables, and so on. <strong>Note:</strong> If the web service operation is marked as one-way, the clientgen Ant task never generates the asynchronous flavor of the stub, even if you explicitly set the generateAsyncMethods attribute to True. Valid values for this attribute are True or False. The default value is True, which means the asynchronous methods are generated by default.</td>
<td>Boolean</td>
<td>No</td>
<td>JAX-RPC</td>
</tr>
</tbody>
</table>
Table 2-2  (Cont.) WebLogic-specific Attributes of the clientgen Ant Task

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
<th>Data Type</th>
<th>Required?</th>
<th>JAX-WS, JAX-RPC, or Both?</th>
</tr>
</thead>
</table>
| generatePolicyMethods      | Specifies whether the clientgen Ant task should include WS-Policy-loading methods in the generated stubs. These methods can be used by client applications to load a local WS-Policy file.  
If you specify True, four flavors of a method called getXXXSoapPort() are added as extensions to the Service interface in the generated client stubs, where XXX refers to the name of the web service. Client applications can use these methods to load and apply local WS-Policy files, rather than apply any WS-Policy files deployed with the web service itself. Client applications can specify whether the local WS-Policy file applies to inbound, outbound, or both SOAP messages and whether to load the local WS-Policy from an InputStream or a URI.  
Valid values for this attribute are True or False. The default value is False, which means the additional methods are not generated.  
<p>| getRuntimeCatalog          | Specifies whether the clientgen Ant task should generate the XML catalog artifacts in the client runtime environment. To disable their generation, set this flag to false. This value defaults to true. See Disabling XML Catalogs in the Client Runtime in Developing JAX-WS Web Services for Oracle WebLogic Server. | Boolean   | No        | JAX-WS       |</p>
<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
<th>Data Type</th>
<th>Required?</th>
<th>JAX-WS, JAX-RPC, or Both?</th>
</tr>
</thead>
</table>
| handlerChainFile       | Specifies the name of the XML file that describes the client-side SOAP message handlers that execute when a client application invokes a web service. Each handler specified in the file executes twice:  
  • Directly before the client application sends the SOAP request to the web service  
  • Directly after the client application receives the SOAP response from the web service  
  If you do not specify this clientgen attribute, then no client-side handlers execute, even if they are in your CLASSPATH.  
  See Creating and Using Client-Side SOAP Message Handlers for details and examples about creating client-side SOAP message handlers. | String     | No        | JAX-RPC                               |
| includeGlobalTypes     | Specifies that the clientgen Ant task should generate Java representations of all XML Schema data types in the WSDL, rather than just the data types that are explicitly used in the web service operations.  
  Valid values for this attribute are True or False. The default value is False, which means that clientgen generates Java representations for only the actively-used XML data types. | Boolean    | No        | JAX-RPC                               |
| jaxRPCWrappedArrayStyle| When the clientgen Ant task is generating the Java equivalent to XML Schema data types in the WSDL file, and the task encounters an XML complex type with a single enclosing sequence with a single element with the maxOccurs attribute equal to unbounded, the task generates, by default, a Java structure whose name is the lowest named enclosing complex type or element. To change this behavior so that the task generates a literal array instead, set the jaxRPCWrappedArrayStyle to False.  
  Valid values for this attribute are True or False. The default value is True. | Boolean    | No        | JAX-RPC                               |
### Table 2-2  (Cont.) WebLogic-specific Attributes of the clientgen Ant Task

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
<th>Data Type</th>
<th>Required?</th>
<th>JAX-WS, JAX-RPC, or Both?</th>
</tr>
</thead>
<tbody>
<tr>
<td>packageName</td>
<td>Package name into which the generated client interfaces and stub files are packaged. If you do not specify this attribute, the clientgen Ant task generates Java files whose package name is based on the targetNamespace of the WSDL file. For example, if the targetNamespace is <a href="http://example.org">http://example.org</a>, then the package name might be org.example or something similar. If you want control over the package name, then you should specify this attribute. If you do specify this attribute, Oracle recommends you use all lower-case letters for the package name.</td>
<td>String</td>
<td>No</td>
<td>Both</td>
</tr>
<tr>
<td>serviceName</td>
<td>Name of the web service in the WSDL file for which the corresponding client component files should be generated. The web service name corresponds to the &lt;service&gt; element in the WSDL file. The generated mapping file and client-side copy of the WSDL file will use this name. For example, if you set serviceName to CuteService, the mapping file will be called cuteService_java_wsdl_mapping.xml and the client-side copy of the WSDL will be called CuteService_saved_wsdl.wsdl.</td>
<td>String</td>
<td>This attribute is required only if the WSDL file contains more than one &lt;service&gt; element. The Ant task returns an error if you do not specify this attribute and the WSDL file contains more than one &lt;service&gt; element.</td>
<td>JAX-RPC</td>
</tr>
<tr>
<td>sortSchemaTypes</td>
<td>In an XSD file, two complex types are defined, one a named global type and the other an unnamed local type. By default, clientgen automatically generates its own name for the unnamed local type, and the name generated when compiling different WSDL files is not always consistent. When enabled, the type names in the Java files generated by clientgen will be the same.</td>
<td>Boolean</td>
<td>No</td>
<td>JAX-RPC</td>
</tr>
</tbody>
</table>
Table 2-2  (Cont.) WebLogic-specific Attributes of the clientgen Ant Task

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
<th>Data Type</th>
<th>Required?</th>
<th>JAX-WS, JAX-RPC, or Both?</th>
</tr>
</thead>
<tbody>
<tr>
<td>type</td>
<td>Specifies the type of web service for which you are generating client artifacts: JAX-WS or JAX-RPC. Valid values are: JAXWS, JAXRPC. Default value is JAXRPC.</td>
<td>String</td>
<td>No</td>
<td>Both</td>
</tr>
<tr>
<td>wsd1</td>
<td>Full path name or URL of the WSDL that describes a web service (either WebLogic or non-WebLogic) for which the client component files should be generated. The generated stub factory classes in the client JAR file use the value of this attribute in the default constructor.</td>
<td>String</td>
<td>Yes</td>
<td>Both</td>
</tr>
<tr>
<td>wsd1Location</td>
<td>Specifies the value of the wsd1Location attribute generated on the @WebServiceClient.</td>
<td>String</td>
<td>No</td>
<td>JAX-WS</td>
</tr>
</tbody>
</table>

Examples

The following examples illustrate how to build a clientgen Ant target.

**Example 1 Building a Basic clientgen Ant Target**

In the following example, when the sample build_client target is executed, clientgen uses the WSDL file specified by the wsd1 attribute to generate all the client-side artifacts needed to invoke the web service specified by the serviceName attribute. The clientgen Ant task generates all the artifacts into the /output/clientclasses directory. All generated Java code is in the myapp.myservice.client package. After clientgen has finished, the javac Ant task then compiles the Java code, both clientgen-generated as well as your own client application that uses the generated artifacts and contains your business code. By default, clientgen generates client artifacts based on a JAX-RPC web service.

```xml
<taskdef name="clientgen"
    classname="weblogic.wsee.tools.anttasks.ClientGenTask" />
...
<target name="build_client">
  <clientgen
    wsd1="http://example.com/myapp/myservice.wsdl"
    destDir="/output/clientclasses"
    packageName="myapp.myservice.client"
    serviceName="StockQuoteService" />
  <javac ... />
</target>
```
Example 2 Generating a JAX-WS Web Service Client
In the preceding example, it is assumed that the web service for which you are generating client artifacts is based on JAX-RPC; the following example shows how to use the `type` attribute to specify that the web service is based on JAX-WS:

```xml
<clientgen
type="JAXWS"
wsdl="http://${wls.hostname}:${wls.port}/JaxWsImpl/JaxWsImplService?WSDL"
destDir="/output/clientclasses"
packageName="examples.webservices.jaxws.client"/>
```

Example 3 Compiling and Packaging the Generated Artifacts
If you want the `clientgen` Ant task to compile and package the generated artifacts for you, specify the `destFile` attribute rather than `destDir`. In this example, you do not need to also specify the `javac` Ant task after `clientgen` in the `build.xml` file because the Java code has already been compiled.

```xml
<clientgen
type="JAXWS"
wsdl="http://example.com/myapp/myservice.wsdl"
destFile="/output/jarfiles/myclient.jar"
packageName="myapp.myservice.client"
serviceName="StockQuoteService"/>
```

Example 4 Executing clientgen on a Static WSDL File
You typically execute the `clientgen` Ant task on a WSDL file that is deployed on the Web and accessed using HTTP. Sometimes, however, you might want to execute `clientgen` on a static WSDL file that is packaged in an archive file, such as the WAR or JAR file generated by the `jwsc` Ant task. In this case you must use the following syntax for the `wsdl` attribute:

```
wsdl="jar:file:archive_file!WSDL_file"
```

where `archive_file` refers to the full or relative (to the current directory) name of the archive file and `WSDL_file` refers to the full pathname of the WSDL file, relative to the root directory of the archive file.

The following example shows how to execute `clientgen` on a static WSDL file called `SimpleService.wsdl`, which is packaged in the `WEB-INF` directory of a WAR file called `SimpleImpl.war`, which is located in the `output/myEAR/examples/webservices/simple` sub-directory of the directory that contains the `build.xml` file.

```xml
<clientgen
type="JAXWS"
wsdl="jar:file:output/myEAR/examples/webservices/simple/SimpleImpl.war!/WEB-INF/SimpleService.wsdl"
destDir="/output/clientclasses"
packageName="myapp.myservice.client"/>
```

Example 5 Setting Java Properties
You can use the standard Ant `<sysproperty>` nested element to set Java properties, such as the username and password of a valid WebLogic Server user (if you have enabled access control on the web service) or the name of a client-side trust store that contains trusted certificates, as shown in the following example:
The jwsc Ant task takes as input one or more Java Web Service (JWS) files that contains both standard and WebLogic-specific JWS annotations and generates all the artifacts you need to create a WebLogic web service.

The generated artifacts for **JAX-WS** web services include:

- JSR-109 web service class file at [http://www.jcp.org/en/jsr/detail?id=109](http://www.jcp.org/en/jsr/detail?id=109), such as the service endpoint interface (called JWS\_ClassNamePortType.java, where JWS\_ClassName refers to the JWS class).
- JAXB data binding artifact class file.
- All required deployment descriptors, including:
  - Servlet-based web service deployment descriptor file: web.xml.
  - Ear deployment descriptor files: application.xml and weblogic-application.xml.

**Note:**

For JAX-WS web services:

* The WSDL file is generated when the service endpoint is deployed.
* No EJB deployment descriptors are required for EJB 3.0-based web services.

The generated artifacts for **JAX-RPC** web services include:

- JSR-109 web service class file at [http://www.jcp.org/en/jsr/detail?id=175](http://www.jcp.org/en/jsr/detail?id=175), such as the service endpoint interface (called JWS\_ClassNamePortType.java, where JWS\_ClassName refers to the JWS class).
- All required deployment descriptors, which can include:
- JAX-RPC mapping files.
- Java class-implemented web services: web.xml and weblogic.xml.
- EJB-implemented web services: ejb-jar.xml and weblogic-ejb-jar.xml.
- Ear deployment descriptor files: application.xml and weblogic-application.xml.

- The XML Schema representation of any Java user-defined types used as parameters or return values to the web service operations.
- The WSDL file that publicly describes the web service.

After generating all the artifacts, the jwsc Ant task compiles the Java and JWS files, packages the compiled classes and generated artifacts into a deployable Web application WAR file, and finally creates an exploded Enterprise Application directory that contains the JAR file. You then deploy this Enterprise Application to WebLogic Server.

By default, the jwsc Ant task generates a web service that conforms to the JAX-RPC specification. You can control the type of web services that is generated using the \texttt{type} attribute of the \texttt{<jws>} child element. For example, to generate a JAX-WS web service, set \texttt{type="JAXWS"} attribute of the \texttt{<jws>} child element.

\textbf{Note:}

Although not typical, you can code your JWS file to explicitly implement \texttt{javax.ejb.SessionBean}. See Should You Implement a Stateless Session EJB? in \textit{Developing JAX-WS Web Services for Oracle WebLogic Server} for details. Because this case is not typical, it is assumed in this section that \texttt{jwsc} packages your web service in a Web application WAR file, and EJB-specific information is generated only when necessary.

You specify the JWS file or files you want the jwsc Ant task to compile using the \texttt{<jws>} element, as described in \texttt{jws}. If the \texttt{<jws>} element is an immediate child of the \texttt{jwsc} Ant task, then \texttt{jwsc} generates a separate WAR file for each JWS file. If you want all the JWS files, along with their supporting artifacts, to be packaged in a single WAR file, then group all the \texttt{<jws>} elements under a single \texttt{<module>} element. A single WAR file reduces WebLogic server resources and allows the web services to share common objects, such as user-defined data types. Using this method you can also specify the same context path for the web services; if they are each packaged in their own WAR file then each service must also have a unique context path.

When you use the \texttt{<module>} element, you can use the \texttt{<jwsfileset>} child element to search for a list of JWS files in one or more directories, rather than list each one individually using \texttt{<jws>}.

Typically, \texttt{jwsc} generates a new Enterprise Application exploded directory at the location specified by the \texttt{destDir} attribute. However, if you specify an \textit{existing} Enterprise Application as the destination directory, \texttt{jwsc} updates any existing \texttt{application.xml} file with the new web services information.

Similarly, \texttt{jwsc} typically generates new Web application deployment descriptors (web.xml and weblogic.xml) that describe the generated Web application. If, however, you have an existing Web application to which you want to add web services, you can
use the `<descriptor>` child element of the `<module>` element to specify existing web.xml and weblogic.xml files; in this case, jwsc copies these files to the destDir directory and adds new information to them. Use the standard Ant `<fileset>` element to copy the other existing Web application files to the destDir directory.

**Note:**

The existing web.xml and weblogic.xml files pointed to by the `<descriptor>` element must be XML Schema-based, not DTD-based which will cause the jwsc Ant task to fail with a validation error.

If one or more of the JWS files to be compiled itself includes an invoke of a different web service, then you can use the `<clientgen>` element of jwsc to generate and compile the required client component files, such as the Stub and Service interface implementations for the particular web service you want to invoke. These files are packaged in the generated WAR file so as to make them available to the invoking web service.

The following sections discuss additional important information about jwsc:

- Taskdef Classname
- Child Elements
- Attributes
- Examples

### Taskdef Classname

The following shows the task definition for the jwsc classnamewhich must appear in your Ant build file.

```xml
<taskdef name="jwsc" classname="weblogic.wsee.tools.anttasks.JwscTask"/>
```

### Child Elements

The following shows the child element hierarchy of the jwsc Ant task.

```xml
<jwsc> {1}
    <jws> {0 or more}  
        <WLHttpTransport> {0 or 1}  
        <WLHttpsTransport> {0 or 1}  
        <jmstransportservice> {0 or 1} -- JAX-WS web services only  
        <WLJMSTransport> {0 or 1} -- JAX-RPC web services only  
        <clientgen> {0 or more}  
        <descriptor> {0 or more}
    </jws> {0 or more}  
</jwsc>
```
The `jwsc` Ant task has a variety of attributes and three child elements:

- `<jws>` element—Used as either a child element of `<jwsc>` or `<module>`. Defines the transport (HTTP, HTTPS, or JMS) using on of the following child elements:
  - `<WLHttpTransport>`. See WLHttpTransport.
  - `<WLHttpsTransport>`. See WLHttpsTransport.
  - `<jmstransportservice>` (JAX-WS only). See jmstransportservice.
  - `<WLJMSTransport>` (JAX-RPC only). See jmstransportservice.

For more information, see `jws`.

- `<module>` element—Groups one or more JWS files (also specified with the `<jws>` element) into a single module (WAR file); if you do not specify `<module>`, then each JWS file is packaged into its own module, or WAR file. For more information, see `module`.

- `<binding>` element—Specifies custom binding information. For more information, see `binding`.

The `<clientgen>` and `<descriptor>` elements are children only of the elements that generate modules: either the actual `<module>` element itself, or `<jws>` when used as a child of `<jwsc>`, rather than a child of `<module>`.

The `<jwsfileset>` element can be used only as a child of `<module>`.

The following sections describe each child element in the `jwsc` Ant task in more detail.

### binding

Use the `<binding>` child element to specify one of the following:

- For JAX-WS, one or more customization files that specify JAX-WS and JAXB custom binding declarations. See Customizing XML Schema-to-Java Mapping Using Binding Declarations in Developing JAX-WS Web Services for Oracle WebLogic Server.

- For JAX-RPC, one or more XMLBeans configuration files, which by convention end in `.xsdconfig`. Use this element if your web service uses Apache XMLBeans data types as parameters or return values.

The `<binding>` element is similar to the standard Ant `<Fileset>` element and has all the same attributes. See the Apache Ant documentation on the Fileset element at [http://ant.apache.org/manual/Types/fileset.html](http://ant.apache.org/manual/Types/fileset.html) for the full list of attributes you can specify.

---

**Note:**

The `<binding>` child element is not valid if you specify the `compliedWsdl` attribute of the `<jws>` element.
Use the `<clientgen>` element if the JWS file itself invokes another web service and you want the jwsc Ant task to automatically generate and compile the required client-side artifacts and package them in the Web application WAR file together with the web service. The client-side artifacts include:

- The Java classes or the Stub and Service interface implementations for the particular web service you want to invoke.
- The Java classes for any user-defined XML Schema data types included in the WSDL file.
- For JAX-RPC, the mapping deployment descriptor file which contains information about the mapping between the Java user-defined data types and their corresponding XML Schema types in the WSDL file.

To view this element within the jwsc element hierarchy, see Attributes. See Examples for examples of using the element.

You can specify the standard Ant `<sysproperty>` child element to specify properties required by the web service from which you are generating client-side artifacts. For example, if the web service is secured, you can use the `javax.xml.rpc.security.auth.username|password` properties to set the authenticated username and password. See the Ant documentation at http://ant.apache.org/manual/ for the Java Ant task for additional information about `<sysproperty>`.

You can use the `<clientgen>` child element for generating both JAX-WS and JAX-RPC web services.

The following table describes the attributes of the `<clientgen>` element.
<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
<th>Required</th>
<th>JAX-RPC, JAX-WS, or Both?</th>
</tr>
</thead>
<tbody>
<tr>
<td>autoDetectWrapped</td>
<td>Specifies whether the <code>jwsc</code> Ant task should try to determine whether the parameters and return type of document-literal web services are of type <code>wrapped</code> or <code>bare</code>. When the <code>jwsc</code> Ant task parses a WSDL file to create the stubs, it attempts to determine whether a document-literal web service uses wrapped or bare parameters and return types based on the names of the XML Schema elements, the name of the operations and parameters, and so on. Depending on how the names of these components match up, the <code>jwsc</code> Ant task makes a best guess as to whether the parameters are wrapped or bare. In some cases, however, you might want the Ant task to always assume that the parameters are of type bare; in this case, set the <code>autoDetectWrapped</code> attribute to <code>False</code>. Valid values for this attribute are <code>True</code> or <code>False</code>. The default value is <code>True</code>.</td>
<td>No</td>
<td>JAX-RPC</td>
</tr>
<tr>
<td>catalog</td>
<td>Specifies an external XML catalog file. See Using XML Catalogs in Developing JAX-WS Web Services for Oracle WebLogic Server.</td>
<td>No</td>
<td>JAX-WS</td>
</tr>
<tr>
<td>handlerChainFile</td>
<td>Specifies the name of the XML file that describes the client-side SOAP message handlers that execute when the JWS file invokes a web service. Each handler specified in the file executes twice: • directly before the JWS sends the SOAP request to the invoked web service. • directly after the JWS receives the SOAP response from the invoked web service. If you do not specify this attribute, then no client-side handlers execute when the web service is invoked from the JWS file, even if they are in your CLASSPATH. See Creating and Using Client-Side SOAP Message Handlers in Oracle Fusion Middleware Developing JAX-RPC Web Services for Oracle WebLogic Server for details and examples about creating client-side SOAP message handlers.</td>
<td>No</td>
<td>JAX-RPC</td>
</tr>
</tbody>
</table>
### Table 2-3 (Cont.) Attributes of the `<clientgen>` Element

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
<th>Required</th>
<th>JAX-RPC, JAX-WS, or Both?</th>
</tr>
</thead>
<tbody>
<tr>
<td>generateAsyncMethods</td>
<td>Specifies whether the jwsc Ant task should include methods in the generated stubs that the JWS file can use to invoke a web service operation asynchronously. For example, if you specify True (which is also the default value), and one of the web service operations in the WSDL is called <code>getQuote</code>, then the jwsc Ant task also generates a method called <code>getQuoteAsync</code> in the stubs which the JWS file can use instead of the original <code>getQuote</code> method. This asynchronous flavor of the operation also has an additional parameter, of data type <code>weblogic.wsee.async.AsyncPreCallContext</code>, that the JWS file can use to set asynchronous properties, contextual variables, and so on. <strong>Note:</strong> If the operation of the web service being invoked in the JWS file is marked as one-way, the jwsc Ant task never generates the asynchronous flavor of the stub, even if you explicitly set the <code>generateAsyncMethods</code> attribute to True. Valid values for this attribute are True or False. The default value is True, which means the asynchronous methods are generated by default.</td>
<td>No</td>
<td>JAX-RPC</td>
</tr>
<tr>
<td>Attribute</td>
<td>Description</td>
<td>Required?</td>
<td>JAX-RPC, JAX-WS, or Both?</td>
</tr>
<tr>
<td>--------------------</td>
<td>-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
<td>-----------</td>
<td>----------------------------</td>
</tr>
</tbody>
</table>
| generatePolicyMethods | Specifies whether the jwsc Ant task should include WS-Policy-loading methods in the generated stubs. You can use these methods in your JWS file, when invoking the web service, to load a local WS-Policy file.  
If you specify True, four flavors of a method called getXXXSoapPort() are added as extensions to the Service interface in the generated client stubs, where XXX refers to the name of the web service. You can program the JWS file to use these methods to load and apply local WS-Policy files, rather than apply any WS-Policy file deployed with the web service itself. You can specify in the JWS file whether the local WS-Policy file applies to inbound, outbound, or both SOAP messages and whether to load the local WS-Policy file from an InputStream or a URI.  
Valid values for this attribute are True or False. The default value is False, which means the additional methods are not generated.  
| includeGlobalTypes | Specifies that the jwsc Ant task should generate Java representations of all XML Schema data types in the WSDL, rather than just the data types that are explicitly used in the web service operations.  
Valid values for this attribute are True or False. The default value is False, which means that jwsc generates Java representations for only the actively-used XML data types. | No        | JAX-RPC                     |
| jaxRPCWrappedArrayStyle | When the jwsc Ant task is generating the Java equivalent to XML Schema data types in the WSDL file, and the task encounters an XML complex type with a single enclosing sequence with a single element with the maxOccurs attribute equal to unbounded, the task generates, by default, a Java structure whose name is the lowest named enclosing complex type or element. To change this behavior so that the task generates a literal array instead, set the jaxRPCWrappedArrayStyle to False.  
Valid values for this attribute are True or False. The default value is True. | No        | JAX-RPC                     |
Table 2-3  (Cont.) Attributes of the <clientgen> Element

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
<th>Required?</th>
<th>JAX-RPC, JAX-WS, or Both?</th>
</tr>
</thead>
<tbody>
<tr>
<td>packageName</td>
<td>Package name into which the generated client interfaces and stub files are packaged. Oracle recommends you use all lower-case letters for the package name.</td>
<td>Yes</td>
<td>Both</td>
</tr>
<tr>
<td>serviceName</td>
<td>Name of the web service in the WSDL file for which the corresponding client-side artifacts should be generated. The web service name corresponds to the &lt;service&gt; element in the WSDL file. The generated JAX-RPC mapping file and client-side copy of the WSDL file will use this name. For example, if you set serviceName to CuteService, the JAX-RPC mapping file will be called cuteService_java_wSDL_mapping.xml and the client-side copy of the WSDL will be called CuteService_saved_wSDL.wsdl.</td>
<td>This attribute is required only if the WSDL file contains more than one &lt;service&gt; element. The Ant task returns an error if you do not specify this attribute and the WSDL file contains more than one &lt;service&gt; element.</td>
<td>JAX-RPC</td>
</tr>
<tr>
<td>wsdl</td>
<td>Full path name or URL of the WSDL that describes a web service (either WebLogic or non-WebLogic) for which the client artifacts should be generated. The generated stub factory classes use the value of this attribute in the default constructor.</td>
<td>Yes</td>
<td>Both</td>
</tr>
</tbody>
</table>

descriptor

Use the <descriptor> element to specify that, rather than create new Web application deployment descriptors when generating the WAR that will contain the implementation of the web service, the jwsc task should instead copy existing files and update them with the new information. This is useful when you have an existing Web application to which you want to add one or more web services. You typically use this element together with the standard <Fileset> Ant task to copy other existing Web application artifacts, such as HTML files and Java classes, to the jwsc-generated Web application.

You can use this element with only the following two deployment descriptor files:

- web.xml
- weblogic.xml

Use a separate <descriptor> element for each deployment descriptor file.
The `<descriptor>` element is a child of either `<module>` or `<jws>`, when the latter is a direct child of the main `jwsc` Ant task.

**Note:**

The existing `web.xml` and `weblogic.xml` files pointed to by the `<descriptor>` element must be XML Schema-based, not DTD-based which will cause the `jwsc` Ant task to fail with a validation error.

You can use the `<descriptor>` child element only for generating JAX-RPC web services. To view this element within the `jwsc` element hierarchy, see Attributes. See Examples for examples of using the element.

The following table describes the attributes of the `<descriptor>` element.

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
<th>Required?</th>
</tr>
</thead>
<tbody>
<tr>
<td>file</td>
<td>Full pathname (either absolute or relative to the directory that contains the <code>build.xml</code> file) of the existing deployment descriptor file. The deployment descriptor must be XML Schema-based, not DTD-based. The <code>jwsc</code> Ant task does not update this file directly, but rather, copies it to the newly-generated Web application.</td>
<td>Yes</td>
</tr>
</tbody>
</table>

**jmstransportservice**

**Note:**

You can use the `<jmstransportservice>` child element for configuring SOAP over JMS transport for JAX-WS web services only. For information about configuring SOAP over JMS transport for JAX-RPC web services, see `WLJMSTransport`.

You cannot use SOAP over JMS transport in conjunction with Web Services reliable messaging or streaming SOAP attachments, as described in `Developing JAX-WS Web Services for Oracle WebLogic Server`.

Use the `<jmstransportservice>` element to enable and configure SOAP over JMS transport for JAX-WS web services and clients.

Using SOAP over JMS transport, web services and clients communicate using JMS destinations instead of HTTP connections, offering the following benefits:

- Reliability
- Scalability
- Quality of service
For more information about using SOAP over JMS transport, see Using SOAP Over JMS Transport as the Connection Protocol in Developing JAX-WS Web Services for Oracle WebLogic Server.

The `<jmstransportservice>` element is a child in the `<jws>` element of the `jwsc` Ant task. You can specify zero or one `<jmstransportservice>` element for a given JWS file.

Optionally, you can configure the destination name, destination type, delivery mode, request and response queues, and other JMS transport properties, using the `<jmstransportservice>` element. For a complete list of JMS transport properties supported, see Configuring JMS Transport Properties in Developing JAX-WS Web Services for Oracle WebLogic Server.

The following example shows how to enable and configure JMS transport when generating the web service using `jwsc`.

```xml
<?xml version="1.0"?>
<project name="jaxws.jms.jwsc" default="all">
  <import file="../build-jms.xml"/>
  <path id="client.class.path">
    <pathelement path="${clientclasses.dir}"/>
    <pathelement path="${java.class.path}"/>
  </path>
  <target name="jwsc">
    <jwsc srcdir="." sourcepath="client" destdir="${output.dir}" debug="on" keepGenerated="yes">
      <jws file="JWSCEndpoint.java" type="JAXWS" explode="true">
        <jmstransportservice
          targetService="JWSCEndpointService"
          destinationName="com.oracle.webservices.api.jms.RequestQueue"
          jndiInitialContextFactory="weblogic.jndi.WLInitialContextFactory"
          jndiConnectionFactoryName="weblogic.jms.XAConnectionFactory"
          jndiURL="t3://localhost:7001"
          deliveryMode="PERSISTENT"
          timeToLive="60000"
          priority="1"
          messageType="TEXT"
          activationConfig = "transAttribute=Supports"
        />
      </jws>
    </jwsc>
  </target>
</project>
```

The `<jws>` element specifies the name of a JWS file that implements your web service and for which the Ant task should generate Java code and supporting artifacts and then package into a deployable WAR file inside of an Enterprise Application.

You can specify the `<jws>` element in the following two different levels of the `jwsc` element hierarchy:

- An immediate child element of the `jwsc` Ant task. In this case, `jwsc` generates a separate WAR file for each JWS file. You typically use this method if you are specifying just one JWS file to the `jwsc` Ant task.
- A child element of the `<module>` element, which in turn is a child of `jwsc`. In this case, `jwsc` generates a single WAR file that includes all the generated code and
artifacts for all the JWS files grouped within the <module> element. This method is useful if you want all JWS files to share supporting files, such as common Java data types.

You are required to specify either a <jws> or <module> child element of jwsc.

To view this element within the jwsc element hierarchy, see Attributes. See Examples for examples of using the element.

You can use the standard Ant <FileSet> child element with the <jws> element of jwsc.

You can use the <jws> child element when generating both JAX-WS and JAX-RPC web services.

The following table describes the attributes of the <jws> element. The description specifies whether the attribute applies in the case that <jws> is a child of jwsc, is a child of <module> or in both cases.

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
<th>Required? JAX-RPC, JAX-WS, or Both?</th>
</tr>
</thead>
<tbody>
<tr>
<td>compiledWsd1</td>
<td>Full pathname of the JAR file generated by the wsd1c Ant task based on an existing WSDL file. The JAR file contains the JWS interface file that implements a web service based on this WSDL, as well as data binding artifacts for converting parameter and return value data between its Java and XML representations; the XML Schema section of the WSDL defines the XML representation of the data. You use this attribute only in the “starting from WSDL” use case, in which you first use the wsd1c Ant task to generate the JAR file, along with the JWS file that implements the generated JWS interface. After you update the JWS implementation class with business logic, you run the jwsc Ant task to generate a deployable web service, using the file attribute to specify this updated JWS implementation file. You do not use the compiledWsd1 attribute for the “starting from Java” use case in which you write your JWS file from scratch and the WSDL file that describes the web service is generated by the WebLogic web services runtime. Applies to &lt;jws&gt; when used as a child of both jwsc and &lt;module&gt;.</td>
<td></td>
</tr>
</tbody>
</table>

|
Table 2-5  (Cont.) Attributes of the <jws> Element of the jwsc Ant Task

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
<th>Required?</th>
<th>JAX-RPC, JAX-WS, or Both?</th>
</tr>
</thead>
<tbody>
<tr>
<td>contextPath</td>
<td>Context path (or context root) of the web service. For example, assume the deployed WSDL of a WebLogic web service is as follows: <a href="http://hostname:7001/financial/GetQuote?WSDL">http://hostname:7001/financial/GetQuote?WSDL</a> The context path for this web service is financial. The value of this attribute overrides any other context path set for the JWS file. This includes the transport-related JWS annotations, as well as the transport-related child elements of &lt;jws&gt;. The default value of this attribute is the name of the JWS file, without its extension. For example, if the name of the JWS file is HelloWorldImpl.java, then the default value of its contextPath is HelloWorldImpl. Applies only when &lt;jws&gt; is a direct child of jwsc. For more information about defining the context path, see: • Defining the Context Path of a WebLogic Web Service in Developing JAX-WS Web Services for Oracle WebLogic Server • Defining the Context Path of a WebLogic Web Service in Oracle Fusion Middleware Developing JAX-RPC Web Services for Oracle WebLogic Server.</td>
<td>No</td>
<td>Both</td>
</tr>
<tr>
<td>explode</td>
<td>Specifies whether the generated WAR file that contains the deployable web service is in exploded directory format or not. Valid values for this attribute are true or false. Default value is false, which means that jwsc generates an actual WAR archive file, and not an exploded directory. Applies only when &lt;jws&gt; is a direct child of jwsc.</td>
<td>No</td>
<td>Both</td>
</tr>
<tr>
<td>file</td>
<td>The name of the JWS file that you want to compile. The jwsc Ant task looks for the file in the srcdir directory. Applies to &lt;jws&gt; when used as a child of both jwsc and &lt;module&gt;.</td>
<td>Yes</td>
<td>Both</td>
</tr>
<tr>
<td>generateWsd</td>
<td>Specifies whether the generated WAR file includes the WSDL file in the WEB-INF directory. Valid values for this attribute are true or false. Default value is false, which means that jwsc does not include the WSDL file in the generated WAR file. Applies to &lt;jws&gt; when used as a child of both jwsc and &lt;module&gt;.</td>
<td>Yes</td>
<td>JAX-WS</td>
</tr>
</tbody>
</table>
### Table 2-5  (Cont.) Attributes of the `<jws>` Element of the jwsc Ant Task

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
<th>Required?</th>
<th>JAX-RPC, JAX-WS, or Both?</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>includeSchemas</strong></td>
<td>The full pathname of the XML Schema file that describes an XMLBeans parameter or return value of the web service. To specify more than one XML Schema file, use either a comma or semi-colon as a delimiter: includeSchemas=&quot;po.xsd,customer.xsd&quot;</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>This attribute is only supported in the case where the JWS file explicitly uses an XMLBeans data type as a parameter or return value of a web service operation. If you are not using the XMLBeans data type, the jwsc Ant task returns an error if you specify this attribute.</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Additionally, you can use this attribute only for web services whose SOAP binding is document-literal-bare. Because the default SOAP binding of a WebLogic web service is document-literal-wrapped, the corresponding JWS file must include the following JWS annotation:</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
|               | @SOAPBinding(  
|               |   style=SOAPBinding.Style.DOCUMENT,  
|               |   use=SOAPBinding.Use.LITERAL,  
|               |   parameterStyle=SOAPBinding.ParameterStyle.BARE) |
|               | Applies to `<jws>` when used as a child of both jwsc and `<module>`.                                                                                                                                             |
|               | Note: As of WebLogic Server 9.1, using XMLBeans 1.X data types (in other words, extensions of com.bea.xml.XmlObject) as parameters or return types of a WebLogic web service is deprecated. New applications should use XMLBeans 2.x data types. |
| **name**      | The name of the generated WAR file (or exploded directory, if the explode attribute is set to true) that contains the deployable web service. If an actual JAR archive file is generated, the name of the file will have a .war extension.  |
|               | The default value of this attribute is the name of the JWS file, specified by the file attribute.                                                                                                                                 |
|               | Applies only when `<jws>` is a direct child of jwsc.                                                                                                                                                                               |
| **type**      | Specifies the type of web service to generate: JAX-WS or JAX-RPC.                                                                                                                                                                    |
|               | Valid values are:  
|               | • JAXWS  
|               | • JAXRPC  
|               | Default value is JAXRPC.                                                                                                                                                                                                          | No  | Both             |
Table 2-5  (Cont.) Attributes of the <jws> Element of the jwsc Ant Task

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
<th>Required?</th>
<th>JAX-RPC, JAX-WS, or Both?</th>
</tr>
</thead>
<tbody>
<tr>
<td>wsdlOnly</td>
<td>Specifies that only a WSDL file should be generated for this JWS file. <strong>Note:</strong> Although the other artifacts, such as the deployment descriptors and service endpoint interface, are not generated, data binding artifacts are generated because the WSDL must include the XML Schema that describes the data types of the parameters and return values of the web service operations. The WSDL is generated into the destDir directory. The name of the file is JWS_ClassNameService.wsdl, where JWS_ClassName refers to the name of the JWS class. JWS_ClassNameService is also the name of web service in the generated WSDL file. If you set this attribute to true but also set the explode attribute to false (which is also the default value), then jwsc ignores the explode attribute and always generates the output in exploded format. Valid values for this attribute are true or false. The default value is false, which means that all artifacts are generated by default, not just the WSDL file. Applies only when &lt;jws&gt; is a child of jwsc.</td>
<td>No</td>
<td>Both</td>
</tr>
</tbody>
</table>

`jwsfileset`

Use the `<jwsfileset>` child element of `<module>` to specify one or more directories in which the jwsc Ant task searches for JWS files to compile. The list of JWS files that jwsc finds is then treated as if each file had been individually specified with the `<jws>` child element of `<module>`.

Use the standard nested elements of the `<FileSet>` Ant task to narrow the search. For example, use the `<include>` element to specify the pattern matching that `<jwsfileset>` should follow when determining the JWS files it should include in the list. See the Ant documentation at [http://ant.apache.org/manual/](http://ant.apache.org/manual/) for details about `<FileSet>` and its nested elements.

You can use the `<jwsfileset>` child element for generating both JAX-WS and JAX-RPC web services.

To view this element within the jwsc element hierarchy, see Attributes. See Examples for examples of using the element.

The following table describes the attributes of the `<jwsfileset>` element.
Table 2-6   Attributes of the <jwsfileset> Element

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
<th>Required?</th>
<th>JAX-RPC, JAX-WS, or Both?</th>
</tr>
</thead>
<tbody>
<tr>
<td>scrdir</td>
<td>Specifies the directories (separated by semi-colons) that the jwsc Ant task should search for JWS files to compile.</td>
<td>Yes</td>
<td>Both</td>
</tr>
</tbody>
</table>
| type      | Specifies the type of web service to generate for each found JWS file: JAX-WS or JAX-RPC. Valid values are: • JAXWS • JAXRPC  
Default value is JAXRPC. | No         | Both                      |

module

The <module> element groups one or more <jws> elements together so that their generated code and artifacts are packaged in a single Web application (WAR) file. The <module> element is a child of the main jwsc Ant task.

You can group only web services implemented with the same backend component (Java class or stateless session EJB) under a single <module> element; you cannot mix and match. By default, jwsc always implements your web service as a plain Java class; the only exception is if you have implemented a stateless session EJB in your JWS file. This means, for example, that if one of the JWS files specified by the <jws> child element of <module> implements javax.ejb.SessionBean, then all its sibling <jws> files must also implement javax.ejb.SessionBean. If this is not possible, then you cannot group all the JWS files under a single <module>.

The web services within a module must have the same contextPath, but must have unique serviceURIs. You can set the common contextPath by specifying it as an attribute to the <module> element, or ensuring that the @WLXXXTransport annotations (for JAX-RPC only) and/or <WLXXXTransport> elements for each web service have the same value for the contextPath attribute. The jwsc Ant task validates these values and returns an error if they are not unique. For more information about defining the context path, see:

• Defining the Context Path of a WebLogic Web Service in Developing JAX-WS Web Services for Oracle WebLogic Server
• Defining the Context Path of a WebLogic Web Service in Oracle Fusion Middleware Developing JAX-RPC Web Services for Oracle WebLogic Server.

You must specify at least one <jws> child element of <module>.

You can use the <module> child element when generating both JAX-WS and JAX-RPC web services.

To view this element within the jwsc element hierarchy, see Attributes. See Examples for examples of using the element.

The following table describes the attributes of the <module> element.
Table 2-7  Attributes of the `<module>` Element of the jwsc Ant Task

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
<th>Required?</th>
<th>JAX-RPC, JAX-WS, or Both?</th>
</tr>
</thead>
<tbody>
<tr>
<td>contextPath</td>
<td>Context path (or context root) of all the web services contained in this module. For example, assume the deployed WSDL of a WebLogic web service is as follows: <a href="http://hostname:7001/financial/GetQuote?WSDL">http://hostname:7001/financial/GetQuote?WSDL</a>. The context path for this web service is financial. The value of this attribute overrides any other context path set for any of the JWS files contained in this module. This includes the transport-related JWS annotations, as well as the transport-related child elements of <code>&lt;jws&gt;</code>. The default value of this attribute is the name of the JWS file, without its extension. For example, if the name of the JWS file is HelloWorldImpl.java, then the default value of its contextPath is HelloWorldImpl. For more information about defining the context path, see: • Defining the Context Path of a WebLogic Web Service in Developing JAX-WS Web Services for Oracle WebLogic Server • Defining the Context Path of a WebLogic Web Service in Oracle Fusion Middleware Developing JAX-RPC Web Services for Oracle WebLogic Server.</td>
<td>Only required to ensure that the context paths of multiple web services in a single WAR are the same.</td>
<td>Both</td>
</tr>
<tr>
<td>ejbWsInWar</td>
<td>Specifies whether to package EJB-based web services in a WAR file instead of a JAR file. Valid values for this attribute are true or false. Default value is false, which means that EJB-based web services are packaged in a JAR file.</td>
<td>No</td>
<td>JAX-WS</td>
</tr>
<tr>
<td>explode</td>
<td>Specifies whether the generated WAR file that contains the deployable web service(s) is in exploded directory format or not. Valid values for this attribute are true or false. Default value is false, which means that jwsc generates an actual WAR archive file, and not an exploded directory.</td>
<td>No</td>
<td>Both</td>
</tr>
<tr>
<td>generateWsd1</td>
<td>Specifies whether the generated WAR file includes the WSDL file. Valid values for this attribute are true or false. Default value is false, which means that jwsc generates an actual WAR archive file, and not an exploded directory.</td>
<td>No</td>
<td>JAX-WS</td>
</tr>
<tr>
<td>name</td>
<td>The name of the generated WAR file (or exploded directory, if the explode attribute is set to true) that contains the deployable web service(s). If an actual WAR archive file is generated, the name of the file will have a .war extension. The default value of this attribute is jws.</td>
<td>No</td>
<td>Both</td>
</tr>
</tbody>
</table>
### Table 2-7 (Cont.) Attributes of the `<module>` Element of the jwsc Ant Task

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
<th>Required?</th>
<th>JAX-RPC, JAX-WS, or Both?</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>wsldOnly</code></td>
<td>Specifies that <em>only</em> a WSDL file should be generated for each JWS file specified by the <code>&lt;jws&gt;</code> child element of <code>&lt;module&gt;</code>. Note: Although the other artifacts, such as the deployment descriptors and service endpoint interface, are not generated, data binding artifacts <em>are</em> generated because the WSDL must include the XML Schema that describes the data types of the parameters and return values of the web service operations. The WSDL is generated into the <code>destDir</code> directory. The name of the file is <code>JWS_ClassNameService.wsdl</code>, where <code>JWS_ClassName</code> refers to the name of the JWS class. <code>JWS_ClassNameService</code> is also the name of web service in the generated WSDL file. If you set this attribute to <code>true</code> but also set the <code>explode</code> attribute to <code>false</code> (which is also the default value), then jwsc ignores the <code>explode</code> attribute and always generates the output in exploded format. Valid values for this attribute are <code>true</code> or <code>false</code>. The default value is <code>false</code>, which means that all artifacts are generated by default, not just the WSDL file.</td>
<td>No</td>
<td>Both</td>
</tr>
</tbody>
</table>

### WLHttpTransport

Use the `<WLHttpTransport>` child element of the `<jws>` element to specify the context path and service URI sections of the URL used to invoke the web service over the HTTP transport, as well as the name of the port in the generated WSDL.

You can specify one or zero `<WLHttpTransport>` elements for a given JWS file.

You can use the `<WLHttpTransport>` child element when generating both JAX-WS and JAX-RPC web services.

To view this element within the `jwsc` element hierarchy, see Attributes. See Examples for examples of using the element.

The following table describes the attributes of `<WLHttpTransport>`. 
<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
<th>Required?</th>
<th>JAX-RPC, JAX-WS, or Both?</th>
</tr>
</thead>
<tbody>
<tr>
<td>contextPath</td>
<td>Context path (or context root) of the web service. For example, assume the deployed WSDL of a WebLogic web service is as follows: <a href="http://hostname:7001/financial/GetQuote?WSDL">http://hostname:7001/financial/GetQuote?WSDL</a> The contextPath for this web service is financial. The default value of this attribute is the name of the JWS file, without its extension. For example, if the name of the JWS file is HelloWorldImpl.java, then the default value of its contextPath is HelloWorldImpl. For more information about defining the context path, see: • Defining the Context Path of a WebLogic Web Service in Developing JAX-WS Web Services for Oracle WebLogic Server • Defining the Context Path of a WebLogic Web Service in Oracle Fusion Middleware Developing JAX-RPC Web Services for Oracle WebLogic Server.</td>
<td>No</td>
<td>Both</td>
</tr>
<tr>
<td>serviceUri</td>
<td>Web service URI portion of the URL. For example, assume the deployed WSDL of a WebLogic web service is as follows: <a href="http://hostname:7001/financial/GetQuote?WSDL">http://hostname:7001/financial/GetQuote?WSDL</a> The serviceUri for this web service is GetQuote. For JAX-WS, the default value of this attribute is the serviceName element of the @WebService annotation if specified. Otherwise, the name of the JWS file, without its extension, followed by Service. For example, if the serviceName element of the @WebService annotation is not specified and the name of the JWS file is HelloWorldImpl.java, then the default value of its serviceUri is HelloWorldImplService. For JAX-RPC, the default value of this attribute is the name of the JWS file, without its extension. For example, if the name of the JWS file is HelloWorldImpl.java, then the default value of its serviceUri is HelloWorldImpl.</td>
<td>No</td>
<td>Both</td>
</tr>
<tr>
<td>portName</td>
<td>The name of the port in the generated WSDL. This attribute maps to the name attribute of the &lt;port&gt; element in the WSDL. The default value of this attribute is based on the @javax.jws.WebService annotation of the JWS file. In particular, the default portName is the value of the name attribute of @WebService annotation, plus the actual text SoapPort. For example, if @WebService.name is set to MyService, then the default portName is MyServiceSoapPort.</td>
<td>No</td>
<td>Both</td>
</tr>
</tbody>
</table>
WLHttpsTransport

**Note:**

The `<WLHttpsTransport>` element is deprecated as of version 9.2 of WebLogic Server. You should use the `<WLHttpTransport>` element instead because it now supports both the HTTP and HTTPS protocols. If you want client applications to access the web service using only the HTTPS protocol, then you must specify the `@weblogic.jws.security.UserDataConstraint` JWS annotation in your JWS file.

Use the `WLHttpsTransport` element to specify the context path and service URI sections of the URL used to invoke the web service over the secure HTTPS transport, as well as the name of the port in the generated WSDL.

The `<WLHttpsTransport>` element is a child of the `<jws>` element. You can specify one or zero `<WLHttpsTransport>` elements for a given JWS file. You can use the `<WLHttpsTransport>` child element only for generating JAX-RPC web services.

See Attributes top view where this element fits in the `<jwsc>` element hierarchy.

The following table describes the attributes of `<WLHttpsTransport>`.

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
<th>Required?</th>
</tr>
</thead>
<tbody>
<tr>
<td>contextPath</td>
<td>Context path (or context root) of the web service.</td>
<td>No</td>
</tr>
<tr>
<td></td>
<td>For example, assume the deployed WSDL of a WebLogic web service is as follows:</td>
<td></td>
</tr>
<tr>
<td></td>
<td>The contextPath for this web service is <code>financial</code>.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>The default value of this attribute is the name of the JWS file, without its extension. For example, if the name of the JWS file is <code>HelloWorldImpl.java</code>, then the default value of its contextPath is <code>HelloWorldImpl</code>.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>For more information about defining the context path, see:</td>
<td></td>
</tr>
<tr>
<td></td>
<td>• Defining the Context Path of a WebLogic Web Service in <em>Developing JAX-WS Web Services for Oracle WebLogic Server</em></td>
<td></td>
</tr>
<tr>
<td></td>
<td>• Defining the Context Path of a WebLogic Web Service in <em>Oracle Fusion Middleware Developing JAX-RPC Web Services for Oracle WebLogic Server</em>.</td>
<td></td>
</tr>
</tbody>
</table>
Table 2-9 (Cont.) Attributes of the `<WLHttpsTransport>` Child Element of the `<jws>` Element

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
<th>Required?</th>
</tr>
</thead>
<tbody>
<tr>
<td>serviceUri</td>
<td>Web service URI portion of the URL. For example, assume the deployed WSDL of a WebLogic web service is as follows: <a href="https://hostname:7001/financial/GetQuote?WSDL">https://hostname:7001/financial/GetQuote?WSDL</a></td>
<td>No</td>
</tr>
<tr>
<td>[portName]</td>
<td>The name of the port in the generated WSDL. This attribute maps to the <code>name</code> attribute of the <code>&lt;port&gt;</code> element in the WSDL. The default value of this attribute is based on the <code>@javax.jws.WebService</code> annotation of the JWS file. In particular, the default <code>portName</code> is the value of the <code>name</code> attribute of <code>@WebService</code> annotation, plus the actual text <code>SoapPort</code>. For example, if <code>@WebService.name</code> is set to <code>MyService</code>, then the default <code>portName</code> is <code>MyServiceSoapPort</code>.</td>
<td>No</td>
</tr>
</tbody>
</table>

**Note:**

You can use the `<WLJmsTransport>` child element for configuring SOAP over JMS transport for JAX-RPC web services only. For information about configuring JMS transport for JAX-WS web services, see [jmstransportservice](#).

Use the `WLJMSTransport` element to specify the context path and service URI sections of the URL used to invoke the web service over the JMS transport, as well as the name of the port in the generated WSDL. You also specify the name of the JMS queue and connection factory that you have already configured for JMS transport.

The `<WLJmsTransport>` element is a child of the `<jws>` element. You can specify one or zero `<WLJmsTransport>` elements for a given JWS file.

To view this element within the `jwsc` element hierarchy, see Attributes. See Examples for examples of using the element.

The following table describes the attributes of `<WLJmsTransport>`.

---

**WLJMSTransport**

---
Table 2-10  Attributes of the <WLJMSTransport> Child Element of the <jws> Element

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
<th>Required?</th>
</tr>
</thead>
<tbody>
<tr>
<td>contextPath</td>
<td>Context path (or context root) of the web service. For example, assume the deployed WSDL of a WebLogic web service is as follows: <code>http://hostname:7001/financial/GetQuote?WSDL</code> The contextPath for this web service is financial. The default value of this attribute is the name of the JWS file, without its extension. For example, if the name of the JWS file is <code>HelloWorldImpl.java</code>, then the default value of its contextPath is HelloWorldImpl. For more information about defining the context path, see: • Defining the Context Path of a WebLogic Web Service in Developing JAX-WS Web Services for Oracle WebLogic Server • Defining the Context Path of a WebLogic Web Service in Oracle Fusion Middleware Developing JAX-RPC Web Services for Oracle WebLogic Server.</td>
<td>No</td>
</tr>
<tr>
<td>serviceUri</td>
<td>Web service URI portion of the URL. For example, assume the deployed WSDL of a WebLogic web service is as follows: <code>http://hostname:7001/financial/GetQuote?WSDL</code> The serviceUri for this web service is GetQuote. For JAX-WS, the default value of this attribute is the serviceName element of the @WebService annotation if specified. Otherwise, the name of the JWS file, without its extension, followed by Service. For example, if the serviceName element of the @WebService annotation is not specified and the name of the JWS file is <code>HelloWorldImpl.java</code>, then the default value of its serviceUri is HelloWorldImplService. For JAX-RPC, the default value of this attribute is the name of the JWS file, without its extension. For example, if the name of the JWS file is <code>HelloWorldImpl.java</code>, then the default value of its serviceUri is HelloWorldImpl.</td>
<td>No</td>
</tr>
<tr>
<td>portName</td>
<td>The name of the port in the generated WSDL. This attribute maps to the name attribute of the &lt;port&gt; element in the WSDL. The default value of this attribute is based on the @javax.jws.WebService annotation of the JWS file. In particular, the default portName is the value of the name attribute of @WebService annotation, plus the actual text SoapPort. For example, if @WebService.name is set to MyService, then the default portName is MyServiceSoapPort.</td>
<td>No</td>
</tr>
<tr>
<td>queue</td>
<td>The JNDI name of the JMS queue that you have configured for the JMS transport. See Using JMS Transport as the Connection Protocol in Oracle Fusion Middleware Developing JAX-RPC Web Services for Oracle WebLogic Server for details about using JMS transport. The default value of this attribute, if you do not specify it, is <code>weblogic.wsee.DefaultQueue</code>. You must still create this JMS queue in the WebLogic Server instance to which you deploy your web service.</td>
<td>No</td>
</tr>
</tbody>
</table>
Table 2-10  (Cont.) Attributes of the <WLJMS Transport> Child Element of the <jws> Element

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
<th>Required?</th>
</tr>
</thead>
<tbody>
<tr>
<td>connectionFactory</td>
<td>The JNDI name of the JMS connection factory that you have configured for the JMS transport. The default value of this attribute is the default JMS connection factory for your WebLogic Server instance.</td>
<td>No</td>
</tr>
</tbody>
</table>

Attributes

The following sections describe the attributes of the jwsc Ant task.

- WebLogic-Specific jwsc Attributes
- Standard Ant Attributes and Child Elements That Apply to jwsc

WebLogic-Specific jwsc Attributes

The following table summarizes the WebLogic-specific jwsc attributes.

Table 2-11  Attributes of the jwsc Ant Task

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
<th>Required?</th>
<th>JAX-RPC, JAX-WS, or Both?</th>
</tr>
</thead>
<tbody>
<tr>
<td>applicationXml</td>
<td>Specifies the full name and path of the application.xml deployment descriptor of the Enterprise Application. If you specify an existing file, the jwsc Ant task updates it to include the web services information. However, jwsc does not automatically copy the updated application.xml file to the destDir; you must manually copy this file to the destDIR. If the file does not exist, jwsc creates it. The jwsc Ant task also creates or updates the corresponding weblogic-application.xml file in the same directory. If you do not specify this attribute, jwsc creates or updates the file destDir/META-INF/application.xml, where destDir is the jwsc attribute.</td>
<td>No</td>
<td>Both</td>
</tr>
<tr>
<td>destdir</td>
<td>The full pathname of the directory that will contain the compiled JWS files, XML Schemas, WSDL, and generated deployment descriptor files, all packaged into a JAR or WAR file. The jwsc Ant task creates an exploded Enterprise Application at the specified directory, or updates one if you point to an existing application directory. The jwsc task generates the JAR or WAR file that implements the web service in this directory, as well as other needed files, such as the application.xml file in the META-INF directory; the jwsc Ant task updates an existing application.xml file if it finds one, or creates a new one if not. Use the applicationXML attribute to specify a different application.xml from the default.</td>
<td>Yes</td>
<td>Both</td>
</tr>
</tbody>
</table>
Table 2-11  (Cont.) Attributes of the jwsc Ant Task

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
<th>Required?</th>
<th>JAX-RPC, JAX-WS, or Both?</th>
</tr>
</thead>
<tbody>
<tr>
<td>destEncoding</td>
<td>Specifies the character encoding of the output files, such as the deployment descriptors and XML files. Examples of character encodings are SHIFT-JIS and UTF-8. The default value of this attribute is UTF-8.</td>
<td>No</td>
<td>Both</td>
</tr>
<tr>
<td>dotNetStyle</td>
<td>Specifies that the jwsc Ant task should generate a .NET-style web service. In particular, this means that, in the WSDL of the web service, the value of the name attribute of the &lt;part&gt; element that corresponds to the return parameter is parameters rather than returnParameters. This applies only to document-literal-wrapped web services. The valid values for this attribute are true and false. The default value is true, which means .NET-style web service are generated by default.</td>
<td>No</td>
<td>JAX-RPC</td>
</tr>
<tr>
<td>enableAsyncService</td>
<td>Specifies whether the web service is using one or more of the asynchronous features of WebLogic web service: web service reliable messaging, asynchronous request-response, buffering, or conversations. In the case of web service reliable messaging, you must ensure that this attribute is enabled for both the reliable web service and the web service that is invoking the operations reliably. In the case of the other features (conversations, asynchronous request-response, and buffering), the attribute must be enabled only on the client web service. When this attribute is set to true (default value), WebLogic Server automatically deploys internal modules that handle the asynchronous web service features. Therefore, if you are not using any of these features in your web service, consider setting this attribute to false so that WebLogic Server does not waste resources by deploying unneeded internal modules. Valid values for this attribute are true and false. The default value is true. <strong>Note:</strong> This attribute is deprecated as of Version 9.2 of WebLogic Server.</td>
<td>No</td>
<td>Deprecate d attribute so not applicable.</td>
</tr>
<tr>
<td>keepGenerated</td>
<td>Specifies whether the Java source files and artifacts generated by this Ant task should be regenerated if they already exist. If you specify no, new Java source files and artifacts are always generated and any existing artifacts are overwritten. If you specify yes, the Ant task regenerates only those artifacts that have changed, based on the timestamp of any existing artifacts. Valid values for this attribute are yes or no. The default value is no.</td>
<td>No</td>
<td>Both</td>
</tr>
</tbody>
</table>
Table 2-11  (Cont.) Attributes of the jwsc Ant Task

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
<th>Required?</th>
<th>JAX-RPC, JAX-WS, or Both?</th>
</tr>
</thead>
<tbody>
<tr>
<td>sourcepath</td>
<td>The full pathname of top-level directory that contains the Java files referenced by the JWS file, such as JavaBeans used as parameters or user-defined exceptions. The Java files are in sub-directories of the sourcepath directory that correspond to their package names. The sourcepath pathname can be either absolute or relative to the directory which contains the Ant build.xml file. For example, if sourcepath is /src and the JWS file references a JavaBean called MyType.java which is in the webservices.financial package, then this implies that the MyType.java Java file is stored in the /src/webservices/financial directory. The default value of this attribute is the value of the srcdir attribute. This means that, by default, the JWS file and the objects it references are in the same package. If this is not the case, then you should specify the sourcepath accordingly.</td>
<td>No</td>
<td>Both</td>
</tr>
<tr>
<td>srcdir</td>
<td>The full pathname of top-level directory that contains the JWS file you want to compile (specified with the file attribute of the &lt;jws&gt; child element). The JWS file is in sub-directories of the srcdir directory that corresponds to its package name. The srcdir pathname can be either absolute or relative to the directory which contains the Ant build.xml file. For example, if srcdir is /src and the JWS file called MyService.java is in the webservices.financial package, then this implies that the MyService.java JWS file is stored in the /src/webservices/financial directory.</td>
<td>Yes</td>
<td>Both</td>
</tr>
<tr>
<td>srcEncoding</td>
<td>Specifies the character encoding of the input files, such as the JWS file or configuration XML files. Examples of character encodings are SHIFT-JIS and UTF-8. The default value of this attribute is the character encoding set for the JVM.</td>
<td>No</td>
<td>Both</td>
</tr>
</tbody>
</table>

Standard Ant Attributes and Child Elements That Apply to jwsc

In addition to the WebLogic-defined jwsc attributes, you can also define the following standard javac attributes; see the Ant documentation at http://ant.apache.org/manual/ for additional information about each attribute:

- bootclasspath
- bootClasspathRef
- classpath
- classpathRef
- compiler
- debug
• debugLevel
• depend
• deprecation
• destdir
• encoding
• extdirs
• failonerror
• fork
• includeantruntime
• includejavaruntime
• listfiles
• memoryInitialSize
• memoryMaximumSize
• nowarn
• optimize
• proceed
• source
• sourcepath
• sourcepathRef
• tempdir
• verbose

You can also use the following standard Ant child elements with the jwsc Ant task:

• <SourcePath>
• <Classpath>
• <Extdirs>

You can use the following standard Ant elements with the <jws> and <module> child elements of the jwsc Ant task:

• <FileSet>
• <ZipFileSet>

Examples

The following examples illustrate how to build a jwsc Ant target. See Developing JAX-WS Web Services for Oracle WebLogic Server or Oracle Fusion Middleware Developing JAX-RPC Web Services for Oracle WebLogic Server for samples of complete build.xml files that contain many other targets that are useful when iteratively developing a WebLogic web service, such as clean, deploy, client, and run.
Example 1 Building a Basic jwsc Ant Target
The following sample shows a very simple usage of jwsc. In this example, the JWS file called TestServiceImpl.java is located in the src/examples/webservices/jwsc sub-directory of the directory that contains the build.xml file. The jwsc Ant task generates the web service artifacts in the output/TestEar sub-directory. In addition to the web service JAR file, the jwsc Ant task also generates the application.xml file that describes the Enterprise Application in the output/TestEar/META-INF directory.

```xml
<target name="build-service">
  <jwsc
    srcdir="src"
    destdir="output/TestEar">
    <jws file="examples/webservices/jwsc/TestServiceImpl.java" />
  </jwsc>
</target>
```

Example 2 Generating a JAX-WS Web Service
By default, the jwsc Ant task generates a web service that conforms to the JAX-RPC specification. You can control the type of web services that is generated using the type attribute of the <jws> child element as shown in the following example. In this example, a JAX-WS web service is generated.

```xml
<target name="build-service8">
  <jwsc
    srcdir="src"
    destdir="${ear-dir}"
    type="JAXWS">
    <jws file="examples/webservices/jaxws/JaxWsImpl.java" />
  </jwsc>
</target>
```

Example 3 Specifying Multiple JWS Files
This example shows how to specify multiple JWS files, resulting in separate web services packaged in their own Web application WAR files, although all are still deployed as part of the same Enterprise Application. This example also shows how to enable debugging and verbose output; how to specify that jwsc not regenerate any existing temporary files in the output directory; and how to use classpathref attribute to add to the standard CLASSPATH by referencing a path called add.class.path that has been specified elsewhere in the build.xml file using the standard Ant <path> target.

```xml
<path id="add.class.path">
  <pathelement path="${myclasses-dir}"/>
  <pathelement path="${java.class.path}"/>
</path>
...
<target name="build-service2">
  <jwsc
    srcdir="src"
    destdir="output/TestEar"
    verbose="on"
    debug="on"
    keepGenerated="yes"
    classpathref="add.class.path" >
    <jws file="examples/webservices/jwsc/TestServiceImpl.java" type="JAXWS"/>
    <jws file="examples/webservices/jwsc/AnotherTestServiceImpl.java"
```
Example 4 Packaging Multiple Web Services Into a Single WAR File
If you want to package multiple web services into a single WAR file, group the <jws> elements under a <module> element, as shown in the following example. In this case, the three web services are packaged in a WAR file called myJar.war, located at the top level of the Enterprise Application exploded directory. The contextPath attribute of <module> specifies that the context path of all three web services is test; this value overrides any context path specified in a transport annotation of the JWS files.

```xml
<target name="build-service3">
  <jwsc srcdir="src" destdir="output/TestEar">
    <module contextPath="test" name="myJar">
      <jws file="examples/webservices/jwsc/TestServiceImpl.java" type="JAXWS"/>
      <jws file="examples/webservices/jwsc/AnotherTestServiceImpl.java" type="JAXWS"/>
      <jws file="examples/webservices/jwsc/SecondTestServiceImpl.java" type="JAXWS"/>
    </module>
  </jwsc>
</target>
```

Example 5 Configuring Multiple Transports
The following example shows how to specify that the JAX-RPC web service can be invoked using all transports (HTTP/HTTPS/JMS). This example also shows how to use the <clientgen> element to generate and include the client-side artifacts (such as the Stub and Service implementations) of the web service described by http://examples.org/complex/ComplexService?WSDL. This indicates that the TestServiceImpl.java JWS file, in addition to implementing a web service, must also acts as a client to the ComplexService web service and must include Java code to invoke operations of ComplexService.

```xml
<target name="build-service4">
  <jwsc srcdir="src" destdir="output/TestEar">
    <jws file="examples/webservices/jwsc/TestServiceImpl.java">
      <WLHttpTransport contextPath="TestService" serviceUri="TestService" portName="TestServicePortHTTP"/>
      <WLJmsTransport contextPath="TestService" serviceUri="JMSTestService" portName="TestServicePortJMS" queue="JMSTransportQueue"/>
      <clientgen wsdl="http://examples.org/complex/ComplexService?WSDL" serviceName="ComplexService" packageName="examples.webservices.simple_client"/>
    </jws>
  </jwsc>
</target>
```
Example 6 Grouping Multiple <jws> Elements into a <module> Element

The following example is very similar to the preceding one, except that it groups the <jws> elements under a <module> element.

In this example, the individual transport elements no longer define their own contextPath attributes; rather, the parent <module> element defines it instead. This improves maintenance and understanding of what jwsc actually does. Also note that the <clientgen> element is a child of <module>, and not <jws> as in the previous example.

```
<target name="build-service5">
  <jwsc
    srcdir="src"
    destdir="output/TestEar">
    <module contextPath="TestService" >
      <jws file="examples/webservices/jwsc/TestServiceImpl.java">
        <WLHttpTransport
          serviceUrl="TestService"
          portName="TestServicePort1"/>
      </jws>
      <jws file="examples/webservices/jwsc/AnotherTestServiceImpl.java" />
      <jws file="examples/webservices/jwsc/SecondTestServiceImpl.java" />
      <clientgen
        wsdl="http://examples.org/complex/ComplexService?WSDL"
        serviceName="ComplexService"
        packageName="examples.webservices.simple_client" />
    </module>
  </jwsc>
</target>
```

Example 7 Specifying a File Set

The following example show how to use the <jwsfileset> element.

In this example, jwsc searches for *.java files in the directory src/examples/webservices/jwsc, relative to the directory that contains build.xml, determines which Java files contain JWS annotations, and then processes each file as if it had been specified with a <jws> child element of <module>. The <include> element is a standard Ant element at http://ant.apache.org/manual/, described in the documentation for the standard <FilesSet> task.

```
<target name="build-service6">
  <jwsc
    srcdir="src"
    destdir="output/TestEar">
    <module contextPath="test" name="myJar">
      <jwsfileset srcdir="src/examples/webservices/jwsc">
        <include name="**/*.java" />
      </jwsfileset>
    </module>
  </jwsc>
</target>
```

Example 8 Updating Existing Web Application Deployment Descriptors

The following example shows how to specify that the jwsc Ant task not create new Web application deployment descriptors, but rather, add to existing ones.

In this preceding example, the explode="true" attribute of <module> specifies that the generated Web application should be in exploded directory format, rather than the default WAR archive file. The <descriptor> child elements specify jwsc should copy the existing web.xml and weblogic.xml files, located in the webapp/WEB-INF subdirectory of the directory that contains the build.xml file, to the new Web
application exploded directory, and that new web service information from the specified JWS file should be added to the files, rather than jwsc creating new ones. The example also shows how to use the standard Ant at http://ant.apache.org/manual/ task to copy additional files to the generated WAR file; if any of the copied files are Java files, the jwsc Ant task compiles the files and puts the compiled classes into the classes directory of the Web application.

```xml
<target name="build-service7">
  <jwsc srcdir="src" destdir="output/TestEar">
    <module contextPath="test" name="myJar" explode="true">
      <jws file="examples/webservices/jwsc/AnotherTestServiceImpl.java" />
      <FileSet dir="webapp">
        <include name="**/*.java" />
      </FileSet>
      <descriptor file="webapp/WEB-INF/web.xml" />
      <descriptor file="webapp/WEB-INF/weblogic.xml" />
    </module>
  </jwsc>
</target>
```

You can specify the type attribute for the `<jws>` or `<jwsfileset>` elements.

**wsdlc**

The wsdlc Ant task generates, from an existing WSDL file, a set of artifacts that together provide a partial Java implementation of the web service described by the WSDL file. By specifying the type attribute, you can generate a partial implementation based on either JAX-WS or JAX-RPC.

By default, it is assumed that the WSDL file includes a single `<service>` element from which the wsdlc Ant task generates artifacts. You can, however, use the `srcServiceName` attribute to specify a specific web service, in the case that there is more than one `<service>` element in the WSDL file, or use the `srcPortName` attribute to specify a specific port of a web service in the case that there is more than one `<port>` child element for a given web service.

The wsdlc Ant task generates the following artifacts:

- A JWS interface file—or service endpoint interface—that implements the web service described by the WSDL file. The interface includes full method signatures that implement the web service operations, and JWS annotations (such as `@WebService` and `@SOAPBinding`) that implement other aspects of the web service. You should not modify this file.

- Data binding artifacts used by WebLogic Server to convert between the XML and Java representations of the web service parameters and return values. The XML Schema of the data types is specified in the WSDL, and the Java representation is generated by the wsdlc Ant task. You should not modify this file.

- A JWS file that contains a partial (stubbed-out) implementation of the generated JWS interface. You need to modify this file to include your business code.

- Optional Javadocs for the generated JWS interface.

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

When you code the JWS implementation file, you can also add additional JWS annotations, although you must abide by the following rules:

- The only standard JSR-181 JWS annotations you can include in the JWS implementation file are `@WebService` and `@HandlerChain`, `@SOAPMessageHandler`, and `@SOAPMessageHandlers`. If you specify any other JWS-181 JWS annotations, the `jwsc` Ant task will return an error when you try to compile the JWS file into a web service.

- You cannot attach policies to the web service within the JWS implementation file using the `weblogic.jws.Policy` or `weblogic.jws.Policies` annotations. You can attach policies to the deployed web service using the WebLogic Server Administration Console if there is not a policy already defined in the WSDL.

- Additionally, you can specify only the `serviceName` and `endpointInterface` attributes of the `@WebService` annotation. Use the `serviceName` attribute to specify a different `<service>` WSDL element from the one that the `wsdlc` Ant task used, in the rare case that the WSDL file contains more than one `<service>` element. Use the `endpointInterface` attribute to specify the JWS interface generated by the `wsdlc` Ant task.

- For JAX-RPC web services, you can specify WebLogic-specific JWS annotations, as required. You cannot use any WebLogic-specific JWS annotations in a JAX-WS web service.


After you have coded the JWS file with your business logic, run the `jwsc` Ant task to generate a complete Java implementation of the web service. Use the `compiledWsdl` attribute of `jwsc` to specify the JAR file generated by the `wsdlc` Ant task which contains the JWS interface file and data binding artifacts. By specifying this attribute, the `jwsc` Ant task does not generate a new WSDL file but instead uses the one in the JAR file. Consequently, when you deploy the web service and view its WSDL, the deployed WSDL will look just like the one from which you initially started.

**Note:**

The only potential difference between the original and deployed WSDL is the value of the `location` attribute of the `<address>` element of the port(s) of the web service. The deployed WSDL will specify the actual hostname and URI of the deployed web service, which is most likely different from that of the original WSDL. This difference is to be expected when deploying a real web service based on a static WSDL.

Depending on the type of partial implementation you generate (JAX-WS or JAX-RPC), the Java package name of the generated complex data types differs, as described in the following guidelines:
• For JAX-WS, if you specify the `packageName` attribute, then all artifacts (Java complex data types, JWS interface, and the JWS interface implementation) are generated into this package. If you want to change the package name of the generated Java complex data types in this case, use the `<binding>` child element of the `wsdlc` Ant task to specify a custom binding declarations file. For information about creating a custom binding declarations file, see Using JAXB Data Binding in Developing JAX-WS Web Services for Oracle WebLogic Server.

• For JAX-RPC, if you specify the `packageName` attribute of the `wsdlc` Ant task, only the generated JWS interface and implementation are in this package. The package name of the generated Java complex data types, however, always corresponds to the XSD Schema type namespace, whether you specify the `packageName` attribute or not.

See Creating a web service from a WSDL File in Developing JAX-WS Web Services for Oracle WebLogic Server for a complete example of using the `wsdlc` Ant task in conjunction with `jwsc`.

The following sections discuss additional important information about `wsdlc`:

• Taskdef Classname
• Child Elements
• Attributes
• Example

**Taskdef Classname**

```xml
<taskdef name="wsdlc" classname="weblogic.wsee.tools.anttasks.WsdlcTask"/>
```

**Child Elements**

The `wsdlc` Ant task has the following WebLogic-specific child elements:

• `binding`
• `xmlcatalog`

For a list of elements associated with the standard Ant `javac` task that you can also set for the `wsdlc` Ant task, see Standard Ant javac Attributes That Apply To `wsdlc`.

**binding**

Use the `<binding>` child element to specify one of the following:

• For JAX-WS, one or more customization files that specify JAX-WS and JAXB custom binding declarations. See Customizing XML Schema-to-Java Mapping Using Binding Declarations in Developing JAX-WS Web Services for Oracle WebLogic Server.

• For JAX-RPC, one or more XMLBeans configuration files, which by convention end in `.xsdconfig`. Use this element if your web service uses Apache XMLBeans at [http://xmlbeans.apache.org/](http://xmlbeans.apache.org/) data types as parameters or return values.

The `<binding>` element is similar to the standard Ant `<Fileset>` element and has all the same attributes. See the Apache Ant documentation on the Fileset element at
for the full list of attributes you can specify.

xmlcatalog

The `<xmlcatalog>` child element specifies the ID of an embedded XML catalog. The following shows the element syntax:

```xml
<xmlcatalog refid="id"/>
```

The ID referenced by `<xmlcatalog>` must match the ID of an embedded XML catalog. You embed an XML catalog in the `build.xml` file using the following syntax:

```xml
<xmlcatalog id="id">
  <entity publicid="public_id" location="uri"/>
</xmlcatalog>
```

In the above syntax, `public_id` specifies the public identifier of the original XML resource (WSDL or XSD) and `uri` specifies the replacement XML resource.

The following example shows how to embed an XML catalog and reference it using `wsdlc`. Relevant code lines are shown in **bold**.

```xml
<target name="wsdlc">
  <wsdlc srcWsdl="wsdl_files/TemperatureService.wsdl" 
    destJwsDir="output/compiledWsdl" 
    destImplDir="output/impl" 
    packageName="examples.webservices.wsdlc"
    <xmlcatalog refid="wsimportcatalog"/>
  </wsdlc>
</target>

<xmlcatalog id="wsimportcatalog">
  <entity publicid="http://helloservice.org/types/HelloTypes.xsd" 
    location="${basedir}/HelloTypes.xsd"/>
</xmlcatalog>
```

See Using XML Catalogs in *Developing JAX-WS Web Services for Oracle WebLogic Server*.

Attributes

The table in the following sections describes the attributes of the `wsdlc` Ant task.

- **WebLogic-Specific wsdlc Attributes**
- **Standard Ant javac Attributes That Apply To wsdlc**

**WebLogic-Specific wsdlc Attributes**

The following table describes the WebLogic-specific `wsdlc` attributes.
Table 2-12 WebLogic-specific Attributes of the wsdlc Ant Task

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
<th>Data Type</th>
<th>Required?</th>
<th>JAX-RPC, JAX-WS, or Both?</th>
</tr>
</thead>
<tbody>
<tr>
<td>autoDetectWrapped</td>
<td>Specifies whether the wsdlc Ant task should try to determine whether the parameters and return type of document-literal web services are of type wrapped or bare.</td>
<td>Boolean</td>
<td>No</td>
<td>JAX-RPC</td>
</tr>
<tr>
<td></td>
<td>When the wsdlc Ant task parses a WSDL file to create the partial JWS file that implements the web service, it attempts to determine whether a document-literal web service uses wrapped or bare parameters and return types based on the names of the XML Schema elements, the name of the operations and parameters, and so on. Depending on how the names of these components match up, the wsdlc Ant task makes a best guess as to whether the parameters are wrapped or bare. In some cases, however, you might want the Ant task to always assume that the parameters are of type bare; in this case, set the autoDetectWrapped attribute to False. Valid values for this attribute are True or False. The default value is True.</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>catalog</td>
<td>Specifies an external XML catalog file. See Using XML Catalogs in Developing JAX-WS Web Services for Oracle WebLogic Server.</td>
<td>String</td>
<td>No</td>
<td>Both</td>
</tr>
<tr>
<td>destImplDir</td>
<td>Directory into which the stubbed-out JWS implementation file is generated. The generated JWS file implements the generated JWS interface file (contained within the JAR file). You update this JWS implementation file, adding Java code to the methods so that they behave as you want, then later specify this updated JWS file to the jwsc Ant task to generate a deployable web service.</td>
<td>String</td>
<td>No</td>
<td>Both</td>
</tr>
</tbody>
</table>
Table 2-12 (Cont.) WebLogic-specific Attributes of the wsdlc Ant Task

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
<th>Data Type</th>
<th>Required?</th>
<th>JAX-RPC, JAX-WS, or Both?</th>
</tr>
</thead>
<tbody>
<tr>
<td>destJavadocDir</td>
<td>Directory into which Javadoc that describes the JWS interface is generated. Because you should never unjar or update the generated JAR file that contains the JWS interface file that implements the specified web service, you can get detailed information about the interface file from this generated Javadoc. You can then use this documentation, together with the generated stubbed-out JWS implementation file, to add business logic to the partially generated web service.</td>
<td>String</td>
<td>No</td>
<td>Both</td>
</tr>
<tr>
<td>destJwsDir</td>
<td>Directory into which the JAR file that contains the JWS interface and data binding artifacts should be generated. The name of the generated JAR file is WSDLFile.wsdl.jar, where WSDLFile refers to the root name of the WSDL file. For example, if the name of the WSDL file you specify to the file attribute is MyService.wsdl, then the generated JAR file is MyService_wsdl.jar.</td>
<td>String</td>
<td>Yes</td>
<td>Both</td>
</tr>
<tr>
<td>explode</td>
<td>Specifies whether the generated JAR file that contains the generated JWS interface file and data binding artifacts is in exploded directory format or not. Valid values for this attribute are true or false. Default value is false, which means that wsdlc generates an actual JAR archive file, and not an exploded directory.</td>
<td>Boolean</td>
<td>No</td>
<td>Both</td>
</tr>
<tr>
<td>Attribute</td>
<td>Description</td>
<td>Data Type</td>
<td>Required?</td>
<td>JAX-RPC, JAX-WS, or Both?</td>
</tr>
<tr>
<td>------------------------------</td>
<td>---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
<td>-----------</td>
<td>-----------</td>
<td>----------------------------</td>
</tr>
<tr>
<td>jaxRPCWrappedArrayStyle</td>
<td>When the wsdlc Ant task is generating the Java equivalent to XML Schema data types in the WSDL file, and the task encounters an XML complex type with a single enclosing sequence with a single element with the <code>maxOccurs</code> attribute equal to <code>unbounded</code>, the task generates, by default, a Java structure whose name is the lowest named enclosing complex type or element. To change this behavior so that the task generates a literal array instead, set the <code>jaxRPCWrappedArrayStyle</code> to <code>False</code>. Valid values for this attribute are <code>True</code> or <code>False</code>. The default value is <code>True</code>.</td>
<td>Boolean</td>
<td>No</td>
<td>JAX-RPC</td>
</tr>
<tr>
<td>packageName</td>
<td>Package into which the generated JWS interface and implementation files should be generated. If you do not specify this attribute, the wsdlc Ant task generates a package name based on the <code>targetNamespace</code> of the WSDL.</td>
<td>String</td>
<td>No</td>
<td>Both</td>
</tr>
<tr>
<td>sortSchemaTypes</td>
<td>In an XSD file, two complex types are defined, one a named global type and the other an unnamed local type. By default, clientgen automatically generates its own name for the unnamed local type, and the name generated when compiling different WSDL files is not always consistent. When enabled, the type names in the Java files generated by clientgen will be the same.</td>
<td>Boolean</td>
<td>No</td>
<td>JAX-RPC</td>
</tr>
</tbody>
</table>
### Table 2-12  (Cont.) WebLogic-specific Attributes of the wsdlc Ant Task

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
<th>Data Type</th>
<th>Required?</th>
<th>JAX-RPC, JAX-WS, or Both?</th>
</tr>
</thead>
<tbody>
<tr>
<td>srcBindingName</td>
<td>Name of the WSDL binding from which the JWS interface file should be generated. The <code>wsdlc</code> Ant task runs against the first <code>&lt;service&gt;</code> element it finds in the WSDL file. Therefore, you only need to specify the <code>srcBindingName</code> attribute if there is more than one <code>&lt;binding&gt;</code> element associated with this first <code>&lt;service&gt;</code> element. If the namespace of the binding is the same as the namespace of the service, then you just need to specify the name of the binding for the value of this attribute. For example: <code>srcBindingName=&quot;MyBinding&quot;</code> However, if the namespace of the binding is different from the namespace of the service, then you must also specify the namespace URI, using the following format: <code>srcBindingName=&quot;{URI}BindingName&quot;</code> For example, if the namespace URI of the <code>MyBinding</code> binding is <code>www.examples.org</code>, then you specify the attribute value as follows: <code>srcBindingName=&quot;{www.examples.org}MyBinding&quot;</code> <strong>Note:</strong> This attribute is deprecated as of Version 9.2 of WebLogic Server. Use <code>srcPortName</code> or <code>srcServiceName</code> instead.</td>
<td>String</td>
<td>Only if the WSDL file contains more than one <code>&lt;binding&gt;</code> element</td>
<td>JAX-RPC</td>
</tr>
<tr>
<td>Attribute</td>
<td>Description</td>
<td>Data Type</td>
<td>Required?</td>
<td>JAX-RPC, JAX-WS, or Both?</td>
</tr>
<tr>
<td>-----------------</td>
<td>-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
<td>-----------</td>
<td>-----------</td>
<td>--------------------------</td>
</tr>
<tr>
<td>srcPortName</td>
<td>Name of the WSDL port from which the JWS interface file should be generated. Set the value of this attribute to the value of the name attribute of the &lt;port&gt; element that corresponds to the web service port for which you want to generate a JWS interface file. The &lt;port&gt; element is a child element of the &lt;service&gt; element in the WSDL file. If you do not specify this attribute, wsdlc generates a JWS interface file from the service specified by srcServiceName. <strong>Note:</strong> For JAX-RPC, if you specify this attribute, you cannot also specify srcServiceName.</td>
<td>String</td>
<td>No</td>
<td>Both</td>
</tr>
</tbody>
</table>
Table 2-12  (Cont.) WebLogic-specific Attributes of the wsdlc Ant Task

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
<th>Data Type</th>
<th>Required?</th>
<th>JAX-RPC, JAX-WS, or Both?</th>
</tr>
</thead>
</table>
| srcServiceName | Name of the web service from which the JWS interface file should be generated. Set the value of this attribute to the value of the name attribute of the &lt;service&gt; element that corresponds to the web service for which you want to generate a JWS interface file. The wsdlc Ant task generates a single JWS endpoint interface and data binding JAR file for a given web service. This means that if the &lt;service&gt; element contains more than one &lt;port&gt; element, the following must be true:  
  • The bindings for each port must be the same or equivalent to each other.  
  • The transport for each port must be different. The wsdlc Ant task determines the transport for a port from the address listed in its &lt;address&gt; child element. Because WebLogic web services support only three transports (JMS, HTTP, and HTTPS), this means that there can be at most three &lt;port&gt; child elements for the &lt;service&gt; element specified by this attribute. The generated JWS implementation file will then include the corresponding @WLXXXTransport annotations (for JAX-RPC web services).  
If you do not specify either this or the srcPortName attribute, the WSDL file must include only one &lt;service&gt; element. The wsdlc Ant task generates the JWS interface file and data binding JAR file from this single web service. **Note:** For JAX-RPC, if you specify this attribute, you cannot also specify srcPortName. | String     | No         | Both       |
Table 2-12  (Cont.) WebLogic-specific Attributes of the wsdlc Ant Task

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
<th>Data Type</th>
<th>Required?</th>
<th>JAX-RPC, JAX-WS, or Both?</th>
</tr>
</thead>
<tbody>
<tr>
<td>srcWsdl</td>
<td>Name of the WSDL from which to generate the JAR file that contains the JWS interface and data binding artifacts. The name must include its pathname, either absolute or relative to the directory which contains the Ant build.xml file.</td>
<td>String</td>
<td>Yes</td>
<td>Both</td>
</tr>
<tr>
<td>type</td>
<td>Specifies the type of web service for which you are generating a partial implementation: JAX-WS or JAX-RPC. Valid values are: JAXWS, JAXRPC. Default value is JAXRPC.</td>
<td>String</td>
<td>No</td>
<td>Both</td>
</tr>
<tr>
<td>typeFamily</td>
<td>Specifies the type of data binding classes to generate. Valid values are: TYLAR—Refers to the standard WebLogic web services data binding classes, described in Using JAXB Data Binding in Developing JAX-WS Web Services for Oracle WebLogic Server. XMLBEANS, XMLBEANS_APACHE. Default value is TYLAR. Note: JAXB data binding classes are always generated for a JAX-WS web service.</td>
<td>String</td>
<td>No</td>
<td>JAX-RPC</td>
</tr>
<tr>
<td>wlw81CallbackGen</td>
<td>Specifies whether to generate a WebLogic Workshop 8.1 style callback. Valid values for this attribute are True or False. The default value is False.</td>
<td>Boolean</td>
<td>No</td>
<td>JAX-RPC</td>
</tr>
</tbody>
</table>

Standard Ant javac Attributes That Apply To wsdlc

In addition to the WebLogic-specific wsdlc attributes, you can also define the following standard javac attributes; see the Ant documentation at http://ant.apache.org/manual/ for additional information about each attribute:

- bootclasspath
- bootClasspathRef
- classpath
- classpathRef
• compiler
• debug
• debugLevel
• depend
• deprecation
• destdir
• encoding
• extdirs
• failonerror
• fork
• includeantruntime
• includejavaruntime
• listfiles
• memoryInitialSize
• memoryMaximumSize
• nowarn
• optimize
• proceed
• source
• sourcepath
• sourcepathRef
• tempdir
• verbose

You can also use the following standard Ant child elements with the \texttt{wsdlc} Ant task:

• \texttt{<FileSet>}
• \texttt{<SourcePath>}
• \texttt{<Classpath>}
• \texttt{<Extdirs>}

**Example**

The following excerpt from an Ant build.xml file shows how to use the \texttt{wsdlc} and \texttt{jwsc} Ant tasks together to build a WebLogic web service. The build file includes two different targets: \texttt{generate-from-wsdl} that runs the \texttt{wsdlc} Ant task against an existing WSDL file, and \texttt{build-service} that runs the \texttt{jwsc} Ant task to build a deployable web service from the artifacts generated by the \texttt{wsdlc} Ant task:

```xml
<taskdef name="wsdlc"
    classname="weblogic.wsee.tools.anttasks.WsdlcTask"/>
<taskdef name="jwsc"
    classname="weblogic.wsee.tools.anttasks.JwscTask"/>
```
In the example, the `wsdlc` Ant task takes as input the `TemperatureService.wsdl` file and generates the JAR file that contains the JWS interface and data binding artifacts into the directory `output/compiledWsdl`. The name of the JAR file is `TemperatureService_wsdl.jar`. The Ant task also generates a JWS file that contains a stubbed-out implementation of the JWS interface into the `output/impl/examples/webservices/wsdlc` directory (a combination of the value of the `destImplDir` attribute and the directory hierarchy corresponding to the specified `packageName`).

For JAX-WS, the name of the stubbed-out JWS implementation file is based on the name of the `<service>` element and its inner `<port>` element in the WSDL file. For example, if the service name is `TemperatureService` and the port name is `TemperaturePortType`, then the generated JWS implementation file is called `TemperatureService_TemperaturePortTypeImpl.java`.

For JAX-RPC, the name of the stubbed-out JWS implementation file is based on the name of the `<portType>` element that corresponds to the first `<service>` element. For example, if the `portType` name is `TemperaturePortType`, then the generated JWS implementation file is called `TemperaturePortTypeImpl.java`.

After running `wsdlc`, you code the stubbed-out JWS implementation file, adding your business logic. Typically, you move this JWS file from the `wsdlc`-output directory to a more permanent directory that contains your application source code; in the example, the fully coded `TemperatureService_TemperaturePortTypeImpl.java` JWS file has been moved to the directory `src/examples/webservices/wsdlc/`. You then run the `jwsc` Ant task, specifying this JWS file as usual. The only additional attribute you must specify is `compiledWsd1` to point to the JAR file generated by the `wsdlc` Ant task, as shown in the preceding example. This indicates that you do not want the `jwsc` Ant task to generate a new WSDL file, because you want to use the original one that has been compiled into the JAR file.

The `wsdlget` Ant task downloads to the local directory a WSDL and its imported XML resources.

You may wish to use the download files when defining and referencing an XML catalog to redirect remote XML resources in your application to a local version of the resources.
See Using XML Catalogs in *Developing JAX-WS Web Services for Oracle WebLogic Server*.

The following sections discuss additional important information about `wsdlget`:

- **Taskdef Classname**
- **Child Elements**
- **Attributes**
- **Example**

### Taskdef Classname

```xml
<taskdef name="wsdlget" class="weblogic.wsee.tools.anttasks.WsdlGetTask"/>
```

### Child Elements

The `wsdlget` Ant task has one WebLogic-specific child element: `<xmlcatalog>`. The `<xmlcatalog>` child element specifies the ID of an embedded XML catalog. The following shows the element syntax:

```xml
<xmlcatalog refid="id"/>
```

The ID referenced by `<xmlcatalog>` must match the ID of an embedded XML catalog. You embed an XML catalog in the `build.xml` file using the following syntax:

```xml
<xmlcatalog id="id">
    <entity publicid="public_id" location="uri"/>
</xmlcatalog>
```

In the above syntax, `public_id` specifies the public identifier of the original XML resource (WSDL or XSD) and `uri` specifies the replacement XML resource.

The following example shows how to embed an XML catalog and reference it using `wsdlget`. Relevant code lines are shown in **bold**.

```xml
<target name="wsdlget">
    <wsdlget
        wsdl="${wsdl}"
        destDir="${wsdl.dir}"
        catalog="wsdlcatalog.xml"/
        <xmlcatalog refid="wsimportcatalog"/>
    </wsdlget>
</target>
```

```xml
<xmlcatalog id="wsimportcatalog">
    <entity publicid="http://helloservice.org/types/HelloTypes.xsd"
        location="${basedir}/HelloTypes.xsd"/>
</xmlcatalog>
```

See Using XML Catalogs in *Developing JAX-WS Web Services for Oracle WebLogic Server*.

### Attributes

The following table describes the attributes of the `wsdlget` Ant task.
<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
<th>Data Type</th>
<th>Required?</th>
<th>JAX-RPC, JAX-WS, or Both?</th>
</tr>
</thead>
<tbody>
<tr>
<td>catalog</td>
<td>Specifies an external XML catalog file. See Using XML Catalogs in Developing JAX-WS Web Services for Oracle WebLogic Server.</td>
<td>String</td>
<td>No</td>
<td>Both</td>
</tr>
<tr>
<td>destDir</td>
<td>Directory into which the XML resources are copied. The generated JWS file implements the generated JWS interface file (contained within the JAR file). You update this JWS implementation file, adding Java code to the methods so that they behave as you want, then later specify this updated JWS file to the jwsc Ant task to generate a deployable web service.</td>
<td>String</td>
<td>Yes</td>
<td>Both</td>
</tr>
<tr>
<td>wsdl</td>
<td>Name of the WSDL to copy to the local directory.</td>
<td>String</td>
<td>No</td>
<td>Both</td>
</tr>
</tbody>
</table>

**Example**

The following excerpt from an Ant build.xml file shows how to use the `wsdlget` Ant task to download a WSDL and its imported XML resources. The XML resources will be saved to the `wsdl` folder in the directory from which the Ant task is run.

```xml
<target name="wsdlget">
  <wsdlget
    wsdl="http://host/service?wsdl"
    destDir="./wsdl/">
  </wsdlget>
</target>
```
JWS Annotation Reference

The WebLogic web services programming model uses the JDK 5.0 metadata annotations feature, specified by JSR-175: A Metadata Facility for the Java™ Programming Language, to provide a set of WebLogic-specific JWS annotations.

- Overview of JWS Annotation Tags
- Web Services Metadata Annotations (JSR-181)
- JAX-WS Annotations (JSR-224)
- JAXB Annotations (JSR-222)
- Common Annotations (JSR-250)
- WebLogic-Specific Annotations

Overview of JWS Annotation Tags

In the metadata annotations programming model, you create an annotated Java file and then use Ant tasks to compile the file into the Java source code and generate all the associated artifacts.

The Java Web Service (JWS) annotated file is the core of your Web Service. It contains the Java code that determines how your Web Service behaves. A JWS file is an ordinary Java class file that uses annotations to specify the shape and characteristics of the Web Service.

The JWS annotations that are supported vary based on whether you are creating a JAX-WS or JAX-RPC Web Service. The following table compares the Web Service annotation support for JAX-WS and JAX-RPC.

<table>
<thead>
<tr>
<th>Annotations</th>
<th>JAX-WS</th>
<th>JAX-RPC</th>
</tr>
</thead>
<tbody>
<tr>
<td>Web Services Metadata Annotations (JSR-181)</td>
<td>Y</td>
<td>Y</td>
</tr>
<tr>
<td>JAX-WS Annotations (JSR-224)</td>
<td>Y</td>
<td>N</td>
</tr>
<tr>
<td>JAXB Annotations (JSR-222)</td>
<td>Y</td>
<td>N</td>
</tr>
<tr>
<td>Common Annotations (JSR-250)</td>
<td>Y</td>
<td>N</td>
</tr>
<tr>
<td>WebLogic-Specific Annotations</td>
<td>Y</td>
<td>Y</td>
</tr>
</tbody>
</table>

You can target a JWS annotation at either the class-, method- or parameter-level in a JWS file. Some annotations can be targeted at more than one level, such as `@SecurityRoles` that can be targeted at both the class and method level.

The following example shows a simple JWS file that uses standard JSR-181, shown in bold:
package examples.webservices.complex;

// Import the standard JWS annotation interfaces
import javax.jws.WebMethod;
import javax.jws.WebParam;
import javax.jws.WebResult;
import javax.jws.WebService;
import javax.jws.soap.SOAPBinding;

// Import the BasicStruct JavaBean
import examples.webservices.complex.BasicStruct;

// Standard JWS annotation that specifies that the portType name of the Web
// Service is "ComplexPortType", its public service name is "ComplexService",
// and the targetNamespace used in the generated WSDL is "http://example.org"
@WebService(serviceName="ComplexService", name="ComplexPortType",
    targetNamespace="http://example.org")

// Standard JWS annotation that specifies this is a document-literal-wrapped
// Web Service
@SOAPBinding(style=SOAPBinding.Style.DOCUMENT,
    use=SOAPBinding.Use.LITERAL,
    parameterStyle=SOAPBinding.ParameterStyle.WRAPPED)

/**
* This JWS file forms the basis of a WebLogic Web Service. The Web Services
* has two public operations:
*
* - echoInt(int)
* - echoComplexType(BasicStruct)
*
* The Web Service is defined as a "document-literal" service, which means
* that the SOAP messages have a single part referencing an XML Schema element
* that defines the entire body.
*
*/

public class ComplexImpl {

    // Standard JWS annotation that specifies that the method should be exposed
    // as a public operation. Because the annotation does not include the
    // member-value "operationName", the public name of the operation is the
    // same as the method name: echoInt.
    //
    // The WebResult annotation specifies that the name of the result of the
    // operation in the generated WSDL is "IntegerOutput", rather than the
    // default name "return". The WebParam annotation specifies that the input
    // parameter name in the WSDL file is "IntegerInput" rather than the Java
    // name of the parameter, "input".
    @WebMethod()
    @WebResult(name="IntegerOutput",
        targetNamespace="http://example.org/complex")
    public int echoInt(
        @WebParam(name="IntegerInput",
            targetNamespace="http://example.org/complex")
        int input)
    {
        System.out.println("echoInt ©" + input + "© to you too!");
        return input;
    }

    // Standard JWS annotation to expose method "echoStruct" as a public operation
    // called "echoComplexType"
    // The WebResult annotation specifies that the name of the result of the
    // operation in the generated WSDL is "EchoStructReturnMessage",
    // rather than the default name "return".
    @WebMethod(operationName="echoComplexType")
    @WebResult(name="EchoStructReturnMessage",
        targetNamespace="http://example.org/complex")
public BasicStruct echoStruct(BasicStruct struct) {
    System.out.println("echoComplexType called");
    return struct;
}

The following sections describe the JWS annotations that are supported.

**Web Services Metadata Annotations (JSR-181)**

You can use standard JSR-181 annotations in your JWS file to specify the shape and behavior of your web service. The following table lists these annotations, which are available with the `javax.jws` at [http://docs.oracle.com/javaee/7/api/javax/jws/package-summary.html](http://docs.oracle.com/javaee/7/api/javax/jws/package-summary.html) or `javax.jws.soap` package at [http://docs.oracle.com/javaee/7/api/javax/jws/soap/package-summary.html](http://docs.oracle.com/javaee/7/api/javax/jws/soap/package-summary.html) and are described in more detail in the Web Services Metadata for the Java Platform (JSR-181) specification at [http://www.jcp.org/en/jsr/detail?id=181](http://www.jcp.org/en/jsr/detail?id=181).

<table>
<thead>
<tr>
<th>This annotation . . .</th>
<th>Specifies . . .</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>javax.jws.WebService</code></td>
<td>At the class level that the JWS file implements a Web Service. For more information, see Specifying that the JWS File Implements a Web Service (@WebService Annotation) in Developing JAX-WS Web Services for Oracle WebLogic Server or in Oracle Fusion Middleware Developing JAX-RPC Web Services for Oracle WebLogic Server.</td>
</tr>
<tr>
<td><code>javax.jws.WebMethod</code></td>
<td>That a method of the JWS file should be exposed as a public operation of the Web Service. For more information, see Specifying That a JWS Method Be Exposed as a Public Operation (@WebMethod and @OneWay Annotations) in Developing JAX-WS Web Services for Oracle WebLogic Server or Oracle Fusion Middleware Developing JAX-RPC Web Services for Oracle WebLogic Server.</td>
</tr>
<tr>
<td><code>javax.jws.OneWay</code></td>
<td>That an operation not return a value to the calling application. For more information, see Specifying That a JWS Method Be Exposed as a Public Operation (@WebMethod and @OneWay Annotations) in Developing JAX-WS Web Services for Oracle WebLogic Server or Oracle Fusion Middleware Developing JAX-RPC Web Services for Oracle WebLogic Server.</td>
</tr>
<tr>
<td><code>javax.jws.WebParam</code></td>
<td>The mapping between operation input parameters of the Web Service and elements of the generated WSDL file, as well as specify the behavior of the parameter. For more information, see Customizing the Mapping Between Operation Parameters and WSDL Elements (@WebParam Annotation) in Developing JAX-WS Web Services for Oracle WebLogic Server or Oracle Fusion Middleware Developing JAX-RPC Web Services for Oracle WebLogic Server.</td>
</tr>
<tr>
<td><code>javax.jws.WebResult</code></td>
<td>The mapping between the Web Service operation return value and the corresponding element of the generated WSDL file. For more information, see Customizing the Mapping Between the Operation Return Value and a WSDL Element (@WebResult Annotation) in Developing JAX-WS Web Services for Oracle WebLogic Server or Oracle Fusion Middleware Developing JAX-RPC Web Services for Oracle WebLogic Server.</td>
</tr>
</tbody>
</table>
Table 3-2  (Cont.) Standard JSR-181 JWS Annotations

<table>
<thead>
<tr>
<th>This annotation ...</th>
<th>Specifies ...</th>
</tr>
</thead>
<tbody>
<tr>
<td>javax.jws.HandlerChain</td>
<td>An external handler chain. For more information, see Creating and Using SOAP Message Handlers in Developing JAX-WS Web Services for Oracle WebLogic Server or Oracle Fusion Middleware Developing JAX-RPC Web Services for Oracle WebLogic Server.</td>
</tr>
<tr>
<td>javax.jws.SOAPBinding</td>
<td>At the class level the SOAP bindings of the Web Service (such as, document-encoded or document-literal-wrapped). For more information, see Specifying the Mapping of the Web Service to the SOAP Message Protocol (@SOAPBinding Annotation) in Developing JAX-WS Web Services for Oracle WebLogic Server or Oracle Fusion Middleware Developing JAX-RPC Web Services for Oracle WebLogic Server.</td>
</tr>
</tbody>
</table>

JAX-WS Annotations (JSR-224)

You can use JAX-WS (JSR-224) annotations in your JWS file to specify the shape and behavior of your web service. The following table summarizes the ones that you can use, which are available with the javax.xml.ws package at [http://docs.oracle.com/javaee/7/api/javax/xml/ws/package-summary.html](http://docs.oracle.com/javaee/7/api/javax/xml/ws/package-summary.html) and are described in more detail in JAX-WS 2.1 Annotations at [http://jax-ws.java.net/nonav/2.1.4/docs/annotations.html](http://jax-ws.java.net/nonav/2.1.4/docs/annotations.html).

**Note:**

The JAX-WS JWS annotations are relevant to JAX-WS web services only. This section does not apply to JAX-RPC web services.

Table 3-3  JAX-WS (JSR-244) Annotations

<table>
<thead>
<tr>
<th>This annotation ...</th>
<th>Specifies ...</th>
</tr>
</thead>
<tbody>
<tr>
<td>javax.xml.ws.Action</td>
<td>Whether to allow an explicit association of a WS-Addressing Action message addressing property with input, output, and fault messages of the mapped WSDL operation.</td>
</tr>
<tr>
<td>javax.xml.ws.BindingType</td>
<td>The binding to use for a Web Service implementation class. See Specifying the Binding Type to Use for an Endpoint (@BindingType Annotation) in Developing JAX-WS Web Services for Oracle WebLogic Server.</td>
</tr>
<tr>
<td>javax.xml.ws.FaultAction</td>
<td>Whether to allow an explicit association of a WS-Addressing Action message addressing property with the fault messages of the WSDL operation mapped from the exception class. The @FaultAction annotation is used inside an @Action annotation.</td>
</tr>
<tr>
<td>javax.xml.ws.RequestWrapper</td>
<td>The request wrapper bean to be used at runtime for the methods in the endpoint interface.</td>
</tr>
<tr>
<td>javax.xml.ws.ResponseWrapper</td>
<td>The response wrapper bean to be used at runtime for the methods in the endpoint interface.</td>
</tr>
</tbody>
</table>
Table 3-3  (Cont.) JAX-WS (JSR-244) Annotations

<table>
<thead>
<tr>
<th>This annotation...</th>
<th>Specifies...</th>
</tr>
</thead>
<tbody>
<tr>
<td>javax.xml.ws.ServiceMode</td>
<td>Whether a provider implementation works with the entire protocol message or with the payload only.</td>
</tr>
<tr>
<td>javax.xml.ws.WebEndpoint</td>
<td>The getPortName() methods of a generated service interface.</td>
</tr>
<tr>
<td>javax.xml.ws.WebFault</td>
<td>Service-specific exception classes to customize to the local and namespace name of the fault element and the name of the fault bean.</td>
</tr>
<tr>
<td>javax.xml.ws.WebServiceClient</td>
<td>A generated service interface.</td>
</tr>
<tr>
<td>javax.xml.ws.WebServiceProvider</td>
<td>A provider implementation class.</td>
</tr>
<tr>
<td>javax.xml.ws.WebServiceRef</td>
<td>A reference to a Web Service. See Defining a Web Service Reference Using the @WebServiceRef Annotation in Developing JAX-WS Web Services for Oracle WebLogic Server.</td>
</tr>
</tbody>
</table>

JAXB Annotations (JSR-222)

The JAXB (JSR-222) specification defines the JAXB annotations that you can use in your JWS file to specify the shape and behavior of your web service. The JAXB specification is available at http://jcp.org/en/jsr/detail?id=222, and the annotations you can use are summarized in the following table. Each of these annotations is available with the javax.xml.bind.annotation package at http://docs.oracle.com/javaee/7/api/javax/xml/bind/annotation/package-summary.html and are described in more detail in Customizing Java-to-XML Schema Mapping Using JAXB Annotations in Developing JAX-WS Web Services for Oracle WebLogic Server or the JAXB (JSR-222 at http://jcp.org/en/jsr/detail?id=222) specification.

Note:
The JAXB JWS annotations are relevant to JAX-WS Web Services only. This section does not apply to JAX-RPC Web Services.

Table 3-4  JAXB Mapping Annotations (JSR-222)

<table>
<thead>
<tr>
<th>This annotation...</th>
<th>Specifies...</th>
</tr>
</thead>
<tbody>
<tr>
<td>javax.xml.bind.annotation.XmlAccessType</td>
<td>Whether fields or properties are serialized by default. See Specifying Default Serialization of Fields and Properties (@XmlAccessType) in Developing JAX-WS Web Services for Oracle WebLogic Server.</td>
</tr>
<tr>
<td>javax.xml.bind.annotation.XmlAccessorType</td>
<td>That a property contained in a class be mapped to a local element in the XML schema complex type to which the containing class is mapped. See Mapping Properties to Local Elements (@XmlElement) in Developing JAX-WS Web Services for Oracle WebLogic Server.</td>
</tr>
</tbody>
</table>
Table 3-4 (Cont.) JAXB Mapping Annotations (JSR-222)

<table>
<thead>
<tr>
<th>This annotation ...</th>
<th>Specifies ...</th>
</tr>
</thead>
<tbody>
<tr>
<td>javax.xml.bind.annotation.XmlRootElement</td>
<td>That a top-level class be mapped to a global element in the XML schema that is used by the WSDL of the Web Service. See Mapping a Top-level Class to a Global Element (@XmlRootElement) in Developing JAX-WS Web Services for Oracle WebLogic Server.</td>
</tr>
<tr>
<td>javax.xml.bind.annotation.XmlSeeAlso</td>
<td>The other classes to bind when binding the current class. See Binding a Set of Classes (@XmlSeeAlso) in Developing JAX-WS Web Services for Oracle WebLogic Server.</td>
</tr>
<tr>
<td>javax.xml.bind.annotation.XmlType</td>
<td>That a class or enum type be mapped to an XML Schema type. See Mapping a Value Class to a Schema Type (@XmlType) in Developing JAX-WS Web Services for Oracle WebLogic Server.</td>
</tr>
</tbody>
</table>

Common Annotations (JSR-250)

You can use JAX-WS (JSR-250) annotations in your JWS file to specify the shape and behavior of your web service. Each of these annotations are available with the javax.annotation package at http://docs.oracle.com/javaee/7/api/javax/annotation/package-summary.html and are described in more detail in the Common Annotations for the Java Platform (JSR-250) specification at http://jcp.org/en/jsr/detail?id=250.

Table 3-5 Common Annotations (JSR-250)

<table>
<thead>
<tr>
<th>This annotation ...</th>
<th>Specifies ...</th>
</tr>
</thead>
<tbody>
<tr>
<td>javax.annotation.Resource</td>
<td>A resource that is needed by the application. This annotation may be applied to an application component class or to fields or methods of the component class.</td>
</tr>
<tr>
<td>javax.annotation.PostConstruct</td>
<td>A method that needs to be executed after dependency injection is done to perform initialization.</td>
</tr>
<tr>
<td>javax.annotation.PreDestroy</td>
<td>A callback notification on a method to signal that the instance is in the process of being removed by the container.</td>
</tr>
</tbody>
</table>

WebLogic-Specific Annotations

WebLogic web services define a set of JWS annotations that you can use to specify behavior and features in addition to the standard JSR-181 JWS annotations. The following table summarizes the WebLogic-specific annotations and whether they are
supported for JAX-WS or JAX-RPC. (The majority of annotations are supported for JAX-RPC only.) Each annotation is described in more detail in the sections that follow.

### Table 3-6  WebLogic-specific Annotations

<table>
<thead>
<tr>
<th>This annotation . . .</th>
<th>Specifies . . .</th>
<th>JAX-WS, JAX-RPC, or Both?</th>
</tr>
</thead>
<tbody>
<tr>
<td>com.oracle.webservices.api.jms.JMSTransportClient</td>
<td>That the web service client supports SOAP over JMS transport connection protocol.</td>
<td>JAX-WS</td>
</tr>
<tr>
<td>com.oracle.webservices.api.jms.JMSTransportService</td>
<td>That the web service supports SOAP over JMS transport connection protocol.</td>
<td>JAX-WS</td>
</tr>
<tr>
<td>weblogic.jws.AsyncFailure</td>
<td>The method that handles a potential failure when the main JWS file invokes an operation of another Web Service asynchronously.</td>
<td>JAX-RPC</td>
</tr>
<tr>
<td>weblogic.jws.AsyncResponse</td>
<td>The method that handles the response when the main JWS file invokes an operation of another Web Service asynchronously.</td>
<td>JAX-RPC</td>
</tr>
<tr>
<td>weblogic.jws.Binding</td>
<td>Whether the Web Service uses version 1.1 or 1.2 of the Simple Object Access Protocol (SOAP) implementation when accepting or sending SOAP messages.</td>
<td>JAX-RPC</td>
</tr>
</tbody>
</table>
| weblogic.jws.BufferQueue | The JNDI name of the JMS queue to which WebLogic Server stores:  
  • Buffered Web Service operation invocation.  
  • Reliable Web Service operation invocation. | JAX-RPC |
| weblogic.jws.Callback | That the annotated variable is a callback, which means that you can use the variable to send callback events back to the client Web Service that invoked an operation of the target Web Service. | JAX-RPC |
| weblogic.jws.CallbackMethod | The method in the client Web Service that handles the messages it receives from the callback Web Service. | JAX-RPC |
| weblogic.jws.CallbackService | That the JWS file is actually a Java interface that describes a callback Web Service. | JAX-RPC |
| weblogic.jws.Context | That the annotated field provides access to the runtime context of the Web Service. | JAX-RPC |
| weblogic.jws.Conversation | That a method annotated with the @Conversation annotation can be invoked as part of a conversation between two WebLogic Web Services or a stand-alone Java client and a conversational Web Service. | JAX-RPC |
| weblogic.jws.Conversational | That a JWS file implements a conversational Web Service. | JAX-RPC |
| weblogic.jws.FileStore | That the Web Service does not use the default WebLogic Server default filestore to store internal state information, such as conversational state, but rather uses one specified by the programmer. | JAX-RPC |
| weblogic.jws.MessageBuffer | Which public methods of a JWS are buffered. If specified at the class-level, then all public methods are buffered; if you want only a subset of the methods to be buffered, specify the annotation at the appropriate method-level. | JAX-RPC |
| weblogic.jws.Policies | An array of @weblogic.jws.Policy annotations. | Both |
# Table 3-6  (Cont.) WebLogic-specific Annotations

<table>
<thead>
<tr>
<th>This annotation . . .</th>
<th>Specifies . . .</th>
<th>JAX-WS, JAX-RPC, or Both?</th>
</tr>
</thead>
<tbody>
<tr>
<td>weblogic.jws.Policy</td>
<td>That a WS-Policy file, which contains information about digital signatures, encryption, or Web Service reliable messaging, should be applied to the request or response SOAP messages.</td>
<td>Both</td>
</tr>
<tr>
<td>weblogic.jws.ReliabilityBuffer</td>
<td>Reliable messaging properties for an operation of a reliable Web Service, such as the number of times WebLogic Server should attempt to deliver the message from the JMS queue to the Web Service implementation, and the amount of time that the server should wait in between retries.</td>
<td>JAX-RPC</td>
</tr>
<tr>
<td>weblogic.jws.ReliabilityErrorHandler</td>
<td>The method that handles the error that results when a client Web Service invokes a reliable Web Service, but the client does not receive an acknowledgement that the reliable Web Service actually received the message.</td>
<td>JAX-RPC</td>
</tr>
<tr>
<td>weblogic.jws.ServiceClient</td>
<td>That the annotated variable in the JWS file is a stub used to invoke another WebLogic Web Service when using the following features: • Web Service reliable messaging • Asynchronous request-response • Conversations</td>
<td>JAX-RPC</td>
</tr>
<tr>
<td>weblogic.jws.StreamAttachments</td>
<td>That the WebLogic Web Services runtime use streaming APIs when reading the parameters of all methods of the Web Service.</td>
<td>JAX-RPC</td>
</tr>
<tr>
<td>weblogic.jws.Transactional</td>
<td>Whether the annotated operation, or all the operations of the JWS file when the annotation is specified at the class-level, runs or run inside of a transaction.</td>
<td>JAX-RPC</td>
</tr>
<tr>
<td>weblogic.jws.Types</td>
<td>A comma-separated list of fully qualified Java class names of the alternative data types for a return type or parameter.</td>
<td>JAX-RPC</td>
</tr>
<tr>
<td>weblogic.jws.WildcardBinding</td>
<td>The XML Schema data type to which a wildcard class, such as javax.xml.soap.SOAPElement or org.apache.xmlbeans.XmlObject, binds.</td>
<td>JAX-RPC</td>
</tr>
<tr>
<td>weblogic.jws.WildcardBindings</td>
<td>An array of @weblogic.jws.WildcardBinding annotations.</td>
<td>JAX-RPC</td>
</tr>
<tr>
<td>weblogic.jws.WLHttpTransport</td>
<td>The context path and service URI sections of the URL used to invoke the Web Service over the HTTP transport, as well as the name of the port in the generated WSDL.</td>
<td>JAX-RPC</td>
</tr>
<tr>
<td>weblogic.jws.WLHttpsTransport</td>
<td>The context path and service URI sections of the URL used to invoke the Web Service over the HTTPS transport, as well as the name of the port in the generated WSDL.</td>
<td>JAX-RPC</td>
</tr>
<tr>
<td>weblogic.jws.WLJmsTransport</td>
<td>The context path and service URI sections of the URL used to invoke the Web Service over the JMS transport, as well as the name of the port in the generated WSDL.</td>
<td>JAX-RPC</td>
</tr>
<tr>
<td>weblogic.jws.WSDL</td>
<td>Whether to expose the WSDL of a deployed WebLogic Web Service.</td>
<td>JAX-RPC</td>
</tr>
<tr>
<td>weblogic.jws.security.CallbackRolesAllowed</td>
<td>An array of @SecurityRole JWS annotations that list the roles that are allowed to invoke the callback methods of the Web Service.</td>
<td>JAX-RPC</td>
</tr>
</tbody>
</table>
Table 3-6  (Cont.) WebLogic-specific Annotations

<table>
<thead>
<tr>
<th>This annotation . . .</th>
<th>Specifies . . .</th>
<th>JAX-WS, JAX-RPC, or Both?</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>weblogic.jws.security.RolesAllowed</code></td>
<td>Whether to enable basic authentication for a Web Service.</td>
<td>JAX-RPC</td>
</tr>
<tr>
<td><code>weblogic.jws.security.RolesReference</code></td>
<td>The list of role names that reference actual roles that are allowed to invoke the Web Service.</td>
<td>JAX-RPC</td>
</tr>
<tr>
<td><code>weblogic.jws.security.RunAs</code></td>
<td>The role and user identity which actually runs the Web Service in WebLogic Server.</td>
<td>JAX-RPC</td>
</tr>
<tr>
<td><code>weblogic.jws.security.SecurityRole</code></td>
<td>The name of a role that is allowed to invoke the Web Service.</td>
<td>JAX-RPC</td>
</tr>
<tr>
<td><code>weblogic.jws.security.SecurityRoleRef</code></td>
<td>A role name reference that links to an already-specified role that is allowed to invoke the Web Service.</td>
<td>JAX-RPC</td>
</tr>
<tr>
<td><code>weblogic.jws.security.UserDataConstraint</code></td>
<td>Whether the client is required to use the HTTPS transport when invoking the Web Service.</td>
<td>JAX-RPC</td>
</tr>
<tr>
<td><code>weblogic.jws.security.WssConfiguration</code></td>
<td>The name of the Web Service security configuration you want the Web Service to use.</td>
<td>Both</td>
</tr>
<tr>
<td><code>weblogic.jws.soap.SOAPBinding</code></td>
<td>The mapping of a Web Service operation onto the SOAP message protocol.</td>
<td>JAX-RPC</td>
</tr>
<tr>
<td><code>weblogic.jws.security.SecurityRoles (deprecated)</code></td>
<td>The roles that are allowed to access the operations of the Web Service.</td>
<td>JAX-RPC</td>
</tr>
<tr>
<td><code>weblogic.jws.security.SecurityIdentity (deprecated)</code></td>
<td>The identity assumed by the Web Service when it is invoked.</td>
<td>JAX-RPC</td>
</tr>
<tr>
<td><code>weblogic.wsee.jws.jaxws.owsm.Property</code></td>
<td>A policy configuration property override. Use this annotation with the <code>weblogic.wsee.jws.jaxws.owsm.SecurityPolicy</code> annotation to override a configuration property when attaching a policy to a web service client.</td>
<td>JAX-WS</td>
</tr>
<tr>
<td><code>weblogic.wsee.jws.jaxws.owsm.SecurityPolicy</code></td>
<td>That an Oracle Web Services Manager (OWSM) security policy be attached to the web service or client.</td>
<td>JAX-WS</td>
</tr>
<tr>
<td><code>weblogic.wsee.jws.jaxws.owsm.SecurityPolicy</code></td>
<td>That an Oracle Web Services Manager (Oracle WSM) WS-Policy file, which contains information about digital signatures or encryption, should be applied to the request or response SOAP messages.</td>
<td>JAX-WS</td>
</tr>
<tr>
<td><code>weblogic.wsee.wstx.wsat.Transactional</code></td>
<td>Whether the annotated class or method runs inside of a web service atomic transaction.</td>
<td>JAX-WS</td>
</tr>
</tbody>
</table>

com.oracle.webservices.api.jms.JMSTransportClient

**Target:** Class

Enables and configures SOAP over JMS transport for JAX-WS web service clients.
Using SOAP over JMS transport, web services and clients communicate using JMS destinations instead of HTTP connections, offering the following benefits:

- Reliability
- Scalability
- Quality of service

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

**Attributes**

Optionally, you can configure the following JMS transport properties using the `@JMSTransportClient` annotation. For a description of the properties, see Configuring JMS Transport Properties in *Developing JAX-WS Web Services for Oracle WebLogic Server*.

- `destinationName`
- `destinationType`
- `enabled`
- `jmsHeaderProperty`
- `jmsMessageProperty`
- `jndiConnectionFactoryName`
- `jndiContextParameters`
- `jndiInitialContextFactory`
- `jndiURL`
- `messageType`
- `priority`
- `replyToName`
- `targetService`
- `timeToLive`

**Note:**

You cannot use SOAP over JMS transport in conjunction with web services reliable messaging or streaming SOAP attachments, as described in *Developing JAX-WS Web Services for Oracle WebLogic Server*.

**Example**

The following sample snippet shows how to use the `@JMSTransportClient` annotation in a client file to enable SOAP over JMS transport.

```java
import javax.xml.ws.WebServiceClient;
import com.oracle.webservices.api.jms.JMSTransportClient;
```
@WebServiceClient (name = "WarehouseService", targetNamespace = "http://oracle.com/samples/",
    wsdlLocation="WarehouseService.wsdl")
@JMSTransportClient {
    destinationName="myQueue",
    replyToName="myReplyToQueue",
    jndiURL="t3://localhost:7001",
    jndiInitialContextFactory="weblogic.jndi.WLInitialContextFactory",
    jndiConnectionFactoryName="weblogic.jms.ConnectionFactory",
    deliveryMode="PERSISTENT", timeToLive="1000", priority="1",
    messageType="TEXT"
}

public class WarehouseService  extends Service { ... }

com.oracle.webservices.api.jms.JMSTransportService

**Target:** Class

Enables and configures SOAP over JMS transport for JAX-WS web services.

Using SOAP over JMS transport, web services and clients communicate using JMS destinations instead of HTTP connections, offering the following benefits:

- Reliability
- Scalability
- Quality of service

For more information about using SOAP over JMS transport, see Using SOAP Over JMS Transport as the Connection Protocol in Developing JAX-WS Web Services for Oracle WebLogic Server.

**Note:**

SOAP over JMS transport is not compatible with the following web service features: reliable messaging and HTTP transport-specific security.

**Attributes**

Optionally, you can configure JMS transport properties using the @JMSTransportService annotation. For a description of the properties, see Configuring JMS Transport Properties in Developing JAX-WS Web Services for Oracle WebLogic Server.

**Example**

The following sample snippet shows how to use the @JMSTransportService annotation in a JWS file to enable SOAP over JMS transport. The @ActivationConfigProperty is used to set service-side MDB configuration properties.

```java
import javax.jws.WebService;
import com.oracle.webservices.api.jms.JMSTransportService;
import com.sun.xml.ws.binding.SOAPBindingImpl;
import javax.ejb.ActivationConfigProperty;
@WebService(name="NotifyServicePortType", serviceName="NotifyService",

ORACLE
weblogic.jws.AsyncFailure

**Target:** Method

Specifies the method that handles a potential failure when the main JWS file invokes an operation of another Web Service asynchronously.

When you invoke, from within a JWS file, a Web Service operation asynchronously, the response (or exception, in the case of a failure) does not return immediately after the operation invocation, but rather, at some later point in time. Because the operation invocation did not wait for a response, a separate method in the JWS file must handle the response when it does finally return; similarly, another method must handle a potential failure. Use the `@AsyncFailure` annotation to specify the method in the JWS file that will handle the potential failure of an asynchronous operation invocation.

The `@AsyncFailure` annotation takes two parameters: the name of the stub for the Web Service you are invoking and the name of the operation that you are invoking asynchronously. The stub is the one that has been annotated with the `@ServiceClient` annotation.

The method that handles the asynchronous failure must follow these guidelines:

- **Return void.**
- **Be named`onMethodNameAsyncFailure`, where`MethodName` is the name of the method you are invoking asynchronously (with initial letter always capitalized.)**

In the main JWS file, the call to the asynchronous method will look something like:

```java
port.getQuoteAsync (apc, symbol);
```

where `getQuote` is the non-asynchronous name of the method, `apc` is the asynchronous pre-call context, and `symbol` is the usual parameter to the `getQuote` operation.

- **Have two parameters: the asynchronous post-call context (contained in the`weblogic.wsee.async.AsyncPostCallContext` object) and the`Throwable` exception, potentially thrown by the asynchronous operation call.**

Within the method itself you can get more information about the method failure from the context, and query the specific type of exception and act accordingly.

Typically, you always use the `@AsyncFailure` annotation to explicitly specify the method that handles asynchronous operation failures. The only time you would not use this annotation is if you want a single method to handle failures for two or more stubs that invoke different Web Services. In this case, although the stubs connect to
different Web Services, each Web Service must have a similarly named method, because the Web Services runtime relies on the name of the method (onMethodNameAsyncFailure) to determine how to handle the asynchronous failure, rather than the annotation. However, if you always want a one-to-one correspondence between a stub and the method that handles an asynchronous failure from one of the operations, then Oracle recommends that you explicitly use @AsyncFailure.

See Invoking a Web Service Using Asynchronous Request-Response in Oracle Fusion Middleware Developing JAX-RPC Web Services for Oracle WebLogic Server for detailed information and examples of using this annotation.

Attributes

The following table lists the attributes of the @AsyncFailure annotation.

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Data Type</th>
<th>Required?</th>
</tr>
</thead>
<tbody>
<tr>
<td>target</td>
<td>The name of the stub of the Web Service for which you want to invoke an operation asynchronously. The stub is the one that has been annotated with the @ServiceClient field-level annotation.</td>
<td>String</td>
<td>Yes</td>
</tr>
<tr>
<td>operation</td>
<td>The name of the operation that you want to invoke asynchronously. This is the actual name of the operation, as it appears in the WSDL file. When you invoke this operation in the main code of the JWS file, you add Async to its name. For example, if set operation=&quot;getQuote&quot;, then in the JWS file you invoke it asynchronously as follows:</td>
<td>String</td>
<td>Yes</td>
</tr>
</tbody>
</table>

Example

The following sample snippet shows how to use the @AsyncFailure annotation in a JWS file that invokes the operation of another Web Service asynchronously; only the relevant Java code is included:

```java
package examples.webservices.async_req_res;
...
public class StockQuoteClientImpl {
    @ServiceClient(wsdlLocation="http://localhost:7001/async/StockQuote?WSDL",
                   serviceName="StockQuoteService", portName="StockQuote")
    private StockQuotePortType port;
    @WebMethod public void getQuote (String symbol) {
        AsyncPreCallContext apc = AsyncPreCallContextFactory.getAsyncPreCallContext();
        apc.setProperty("symbol", symbol);
        try {
            port.getQuoteAsync (apc, symbol);
            System.out.println("in getQuote method of StockQuoteClient WS");
        }
        catch (RemoteException e) {
            e.printStackTrace();
        }
    }
...
The example shows a stub called port, used to invoke the Web Service located at http://localhost:7001/async/StockQuote. The getQuote operation is invoked asynchronously, and any exception from this invocation is handled by the onGetQuoteAsyncFailure method, as specified by the @AsyncFailure annotation.

weblogic.jws.AsyncResponse

**Target:** Method

Specifies the method that handles the response when the main JWS file invokes an operation of another Web Service asynchronously.

When you invoke, from within a JWS file, a Web Service operation asynchronously, the response does not return immediately after the operation invocation, but rather, at some later point in time. Because the operation invocation did not wait for a response, a separate method in the JWS file must handle the response when it does finally return. Use the @AsyncResponse annotation to specify the method in the JWS file that will handle the response of an asynchronous operation invocation.

The @AsyncResponse annotation takes two parameters: the name of the stub for the Web Service you are invoking and the name of the operation that you are invoking asynchronously. The stub is the one that has been annotated with the @ServiceClient annotation.

The method that handles the asynchronous response must follow these guidelines:

- **Return void.**

- **Be named onMethodNameAsyncResponse, where MethodName is the name of the method you are invoking asynchronously (with initial letter always capitalized.)**

  In the main JWS file, the call to the asynchronous method will look something like:

  ```java
  port.getQuoteAsync (apc, symbol);
  ```

  where getQuote is the non-asynchronous name of the method, apc is the asynchronous pre-call context, and symbol is the usual parameter to the getQuote operation.

- **Have two parameters: the asynchronous post-call context (contained in the weblogic.wsee.async.AsyncPostCallContext object) and the usual return value of the operation.**

  Within the asynchronous-response method itself you add the code to handle the response. You can also get more information about the method invocation from the context.

  Typically, you always use the @AsyncResponse annotation to explicitly specify the method that handles asynchronous operation responses. The only time you would not use this annotation is if you want a single method to handle the response for two or more stubs that invoke different Web Services. In this case, although the stubs connect to different Web Services, each Web Service must have a similarly named
method, because the Web Services runtime relies on the name of the method (onMethodNamesAsyncResponse) to determine how to handle the asynchronous response, rather than the annotation. However, if you always want a one-to-one correspondence between a stub and the method that handles an asynchronous response from one of the operations, then Oracle recommends that you explicitly use @AsyncResponse.

See Invoking a Web Service Using Asynchronous Request-Response in Oracle Fusion Middleware Developing JAX-RPC Web Services for Oracle WebLogic Server for detailed information and examples of using this annotation.

Attributes

Table 3-8  Attributes of the weblogic.jws.AsyncResponse JWS Annotation Tag

| Name      | Description                                                                 | Data Type | Required?
|-----------|------------------------------------------------------------------------------|-----------|-----------
| target    | The name of the stub of the Web Service for which you want to invoke an operation asynchronously. The stub is the one that has been annotated with the @ServiceClient field-level annotation. | String    | Yes       |
| operation | The name of the operation that you want to invoke asynchronously. This is the actual name of the operation, as it appears in the WSDL file. When you invoke this operation in the main code of the JWS file, you add Async to its name. For example, if set operation="getQuote", then in the JWS file you invoke it asynchronously as follows: port.getQuoteAsync (apc, symbol); | String    | Yes       |

Example

The following sample snippet shows how to use the @AsyncResponse annotation in a JWS file that invokes the operation of another Web Service asynchronously; only the relevant Java code is included:

```java
package examples.webservices.async_req_res;
...
public class StockQuoteClientImpl {
    @ServiceClient(wsdlLocation="http://localhost:7001/async/StockQuote?WSDL",
                   serviceName="StockQuoteService", portName="StockQuote")
    private StockQuotePortType port;
    @WebMethod
    public void getQuote (String symbol) {
        AsyncPreCallContext apc = AsyncCallContextFactory.getAsyncPreCallContext();
        apc.setProperty("symbol", symbol);
        try {
            port.getQuoteAsync(apc, symbol );
            System.out.println("in getQuote method of StockQuoteClient WS");
        } catch (RemoteException e) {
            e.printStackTrace();
        }
    }
    ...
    @AsyncResponse(target="port", operation="getQuote")
    public void onGetQuoteAsyncResponse(AsyncPostCallContext apc, int quote) {
```
The example shows a stub called `port`, used to invoke the Web Service located at `http://localhost:7001/async/StockQuote`. The `getQuote` operation is invoked asynchronously, and the response from this invocation is handled by the `onGetQuoteAsyncResponse` method, as specified by the `@AsyncResponse` annotation.

**weblogic.jws.Binding**

**Target:** Class

Specifies whether the Web Service uses version 1.1 or 1.2 of the Simple Object Access Protocol (SOAP) implementation when accepting or sending SOAP messages. By default, WebLogic Web Services use SOAP 1.1.

**Attributes**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Data Type</th>
<th>Required?</th>
</tr>
</thead>
<tbody>
<tr>
<td>value</td>
<td>Specifies the version of SOAP used in the request and response SOAP messages when the Web Service is invoked. Valid values for this attribute are:</td>
<td>enum</td>
<td>No</td>
</tr>
<tr>
<td></td>
<td>• Type.SOAP11</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>• Type.SOAP12</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>The default value is Type.SOAP11.</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Example**

The following example shows how to specify SOAP 1.2; only the relevant code is shown:

```java
package examples.webservices.soap12;
...
import javax.jws.WebMethod;
import javax.jws.WebService;
import weblogic.jws.Binding;
@WebService(name="SOAP12PortType",
            serviceName="SOAP12Service",
            targetNamespace="http://example.org")
@Binding(Binding.Type.SOAP12)
public class SOAP12Impl {
    @WebMethod()
    public String sayHello(String message) {
        ...
    }
}
```
weblogic.jws.BufferQueue

The following sections describe the annotation in detail.

Description

**Target:** Class

Specifies the JNDI name of the JMS queue to which WebLogic Server stores:

- Buffered Web Service operation invocation.
- Reliable Web Service operation invocation.

When used with buffered Web Services, you use this annotation in conjunction with `@MessageBuffer`, which specifies the methods of a JWS that are buffered. When used with reliable Web Services, you use this annotation in conjunction with `@Policy`, which specifies the reliable messaging WS-Policy file associated with the Web Service.

If you have enabled buffering or reliable messaging for a Web Service, but do not specify the `@BufferQueue` annotation, WebLogic Server uses the default Web Services JMS queue (`weblogic.wsee.DefaultQueue`) to store buffered or reliable operation invocations. This JMS queue is also the default queue for the JMS transport features. It is assumed that you have already created this JMS queue if you intend on using it for any of these features.

See Creating Buffered Web Services and Using Web Services Reliable Messaging in Oracle Fusion Middleware Developing JAX-RPC Web Services for Oracle WebLogic Server for detailed information and examples of creating buffered or reliable Web Services.

Attributes

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Data Type</th>
<th>Required?</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>The JNDI name of the JMS queue to which the buffered or reliable operation invocation is queued.</td>
<td>String</td>
<td>Yes</td>
</tr>
</tbody>
</table>

Example

The following example shows a code snippet from a JWS file in which the public operation is buffered and the JMS queue to which WebLogic Server queues the operation invocation is called `my.buffer.queue`; only the relevant Java code is shown:

```java
package examples.webservices.buffered;
...
@WebService(name="BufferedPortType",
    serviceName="BufferedService",
    targetNamespace="http://example.org")
@BufferQueue(name="my.buffer.queue")
public class BufferedImpl {
    ...
    @WebMethod() 
```
@MessageBuffer(retryCount=10, retryDelay="10 seconds")
@Oneway()
public void sayHelloNoReturn(String message) {
    System.out.println("sayHelloNoReturn: " + message);
}

weblogic.jws.Callback

The following sections describe the annotation in detail.

Description

**Target:** Field

Specifies that the annotated variable is a callback, which means that you can use the variable to send callback events back to the client Web Service that invoked an operation of the target Web Service.

You specify the @Callback annotation in the target Web Service so that it can call back to the client Web Service. The data type of the annotated variable is the callback interface.

The callback feature works between two WebLogic Web Services. When you program the feature, however, you create the following three Java files:

- **Callback interface:** Java interface file that defines the callback methods. You do not explicitly implement this file yourself; rather, the jwsc Ant task automatically generates an implementation of the interface. The implementation simply passes a message from the target Web Service back to the client Web Service. The generated Web Service is deployed to the same WebLogic Server that hosts the client Web Service.

- **JWS file that implements the target Web Service:** The target Web Service includes one or more standard operations that invoke a method defined in the callback interface; this method in turn sends a message back to the client Web Service that originally invoked the operation of the target Web Service.

- **JWS file that implements the client Web Service:** The client Web Service invokes an operation of the target Web Service. This Web Service includes one or more methods that specify what the client should do when it receives a callback message back from the target Web Service via a callback method.

See Using Callbacks to Notify Clients of Events in *Oracle Fusion Middleware Developing JAX-RPC Web Services for Oracle WebLogic Server* for additional overview and procedural information about programming callbacks.

The @Callback annotation does not have any attributes.

Example

The following example shows a very simple target Web Service in which a variable called `callback` is annotated with the @Callback annotation. The data type of the variable is `CallbackInterface`; this means a callback Web Service must exist with this name. After the variable is injected with the callback information, you can invoke the callback methods defined in `CallbackInterface`; in the example, the callback method is `callbackOperation()`.
The text in **bold** shows the relevant code:

```java
package examples.webservices.callback;
import weblogic.jws.WLHttpTransport;
import weblogic.jws.Callback;
import javax.jws.WebService;
import javax.jws.WebMethod;
@WebService(name="CallbackPortType",
    serviceName="TargetService",
    targetNamespace="http://examples.org/")
@WLHttpTransport(contextPath="callback",
    serviceUri="TargetService",
    portName="TargetServicePort")
public class TargetServiceImpl {
    @Callback
    CallbackInterface callback;
    @WebMethod
    public void targetOperation (String message) {
        callback.callbackOperation (message);
    }
}
```

weblogic.jws.CallbackMethod

The following sections describe the annotation in detail.

**Description**

**Target:** Method

Specifies the method in the client Web Service that handles the messages it receives from the callback Web Service. Use the attributes to link the callback message handler methods in the client Web Service with the callback method in the callback interface.

The callback feature works between two WebLogic Web Services. When you program the feature, however, you create the following *three* Java files:

- **Callback interface:** Java interface file that defines the callback methods. You do not explicitly implement this file yourself; rather, the jwsc Ant task automatically generates an implementation of the interface. The implementation simply passes a message from the target Web Service back to the client Web Service. The generated Web Service is deployed to the same WebLogic Server that hosts the client Web Service.

- **JWS file that implements the target Web Service:** The target Web Service includes one or more standard operations that invoke a method defined in the callback interface; this method in turn sends a message back to the client Web Service that originally invoked the operation of the target Web Service.

- **JWS file that implements the client Web Service:** The client Web Service invokes an operation of the target Web Service. This Web Service includes one or more methods that specify what the client should do when it receives a callback message back from the target Web Service via a callback method.

See Using Callbacks to Notify Clients of Events in *Oracle Fusion Middleware Developing JAX-RPC Web Services for Oracle WebLogic Server* for additional overview and procedural information about programming callbacks.
Attributes

Table 3-11 Attributes of the weblogic.jws.CallbackMethod JWS Annotation Tag

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Data Type</th>
<th>Required?</th>
</tr>
</thead>
<tbody>
<tr>
<td>operation</td>
<td>Specifies the name of the callback method in the callback interface for which this method will handle callback messages.</td>
<td>String</td>
<td>Yes</td>
</tr>
<tr>
<td>target</td>
<td>Specifies the name of the stub for which you want to receive callbacks. The stub is the one that has been annotated with the @ServiceClient field-level annotation.</td>
<td>String</td>
<td>Yes</td>
</tr>
</tbody>
</table>

Example

The following example shows a method of a client Web Service annotated with the @CallbackMethod annotation. The attributes show that a variable called port must have previously been injected with stub information and that the annotated method will handle messages received from a callback operation called callbackOperation().

```java
@CallbackMethod(target="port", operation="callbackOperation")
@CallbackRolesAllowed(@SecurityRole(role="engineer", mapToPrincipals="shackell"))
public void callbackHandler(String msg) {
    System.out.println (msg);
}
```

weblogic.jws.CallbackService

The following sections describe the annotation in detail.

Description

**Target:** Class

Specifies that the JWS file is actually a Java interface that describes a callback Web Service. This annotation is analogous to the @javax.jws.WebService, but specific to callbacks and with a reduced set of attributes.

The callback feature works between two WebLogic Web Services. When you program the feature, however, you create the following three Java files:

- **Callback interface:** Java interface file that defines the callback methods. You do not explicitly implement this file yourself; rather, the jwsc Ant task automatically generates an implementation of the interface. The implementation simply passes a message from the target Web Service back to the client Web Service. The generated Web Service is deployed to the same WebLogic Server that hosts the client Web Service.

- **JWS file that implements the target Web Service:** The target Web Service includes one or more standard operations that invoke a method defined in the callback interface; this method in turn sends a message back to the client Web Service that originally invoked the operation of the target Web Service.
• **JWS file that implements the client Web Service:** The client Web Service invokes an operation of the target Web Service. This Web Service includes one or more methods that specify what the client should do when it receives a callback message back from the target Web Service via a callback method.

Use the [@CallbackInterface](https://example.com) annotation to specify that the Java file is a callback interface file.

When you program the callback interface, you specify one or more callback methods; as with standard non-callback Web Services, you annotate these methods with the [@javax.jws.WebMethod](https://example.com) annotation to specify that they are Web Service operations. However, contrary to non-callback methods, you never write the actual implementation code for these callback methods; rather, when you compile the client Web Service with the `jwsc` Ant task, the task automatically creates an implementation of the interface and packages it into a Web Service. This generated implementation specifies that the callback methods all do the same thing: send a message from the target Web Service that invokes the callback method back to the client Web Service.

See Using Callbacks to Notify Clients of Events in *Oracle Fusion Middleware Developing JAX-RPC Web Services for Oracle WebLogic Server* for additional overview and procedural information about programming callbacks.

### Attributes

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Data Type</th>
<th>Required?</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>Name of the callback Web Service. Maps to the <code>&lt;wsdl:portType&gt;</code> element in the WSDL file. Default value is the unqualified name of the Java class in the JWS file.</td>
<td>String</td>
<td>No</td>
</tr>
<tr>
<td>serviceName</td>
<td>Service name of the callback Web Service. Maps to the <code>&lt;wsdl:service&gt;</code> element in the WSDL file. Default value is the unqualified name of the Java class in the JWS file, appended with the string <code>Service</code>.</td>
<td>String</td>
<td>No</td>
</tr>
</tbody>
</table>

### Example

The following example shows a very simple callback interface. The resulting callback Web Service has one callback method, `callbackOperation()`.

```java
package examples.webservices.callback;
import weblogic.jws.CallbackService;
import javax.jws.Oneway;
import javax.jws.WebMethod;
@CallbackService
public interface CallbackInterface {
  @WebMethod
  @Oneway
  public void callbackOperation (String msg);
}
```

---

Chapter 3
WebLogic-Specific Annotations

---

3-21
weblogic.jws.Context

The following sections describe the annotation in detail.

Description

**Target:** Field

Specifies that the annotated field provides access to the runtime context of the Web Service.

When a client application invokes a WebLogic Web Service that was implemented with a JWS file, WebLogic Server automatically creates a context that the Web Service can use to access, and sometimes change, runtime information about the service. Much of this information is related to conversations, such as whether the current conversation is finished, the current values of the conversational properties, changing conversational properties at runtime, and so on. Some of the information accessible via the context is more generic, such as the protocol that was used to invoke the Web Service (HTTP/S or JMS), the SOAP headers that were in the SOAP message request, and so on. The data type of the annotation field must be weblogic.wsee.jws.JwsContext, which is a WebLogic Web Service API that includes methods to query the context.

For additional information about using this annotation, see Accessing Runtime Information about a Web Service in *Developing JAX-WS Web Services for Oracle WebLogic Server*.

This annotation does not have any attributes.

Example

The following snippet of a JWS file shows how to use the @Context annotation; only parts of the file are shown, with relevant code in bold:

```java
import weblogic.jws.Context;
import weblogic.wsee.jws.JwsContext;

public class JwsContextImpl {
    @Context
    private JwsContext ctx;
    @WebMethod()
    public String getProtocol() {
        ...
    }
}
```

weblogic.jws.Conversation

Description

**Target:** Method
Specifies that a method annotated with the @Conversation annotation can be invoked as part of a conversation between two WebLogic Web Services or a stand-alone Java client and a conversational Web Service.

The conversational Web Service typically specifies three methods, each annotated with the @Conversation annotation that correspond to the start, continue, and finish phases of a conversation. Use the @Conversational annotation to specify, at the class level, that a Web Service is conversational and to configure properties of the conversation, such as the maximum idle time.

If the conversation is between two Web Services, the client service uses the @ServiceClient annotation to specify the wsdl, service name, and port of the invoked conversational service. In both the service and stand-alone client cases, the client then invokes the start, continue, and finish methods in the appropriate order to conduct a conversation. The only additional requirement to make a Web Service conversational is that it implement java.io.Serializable.

See Creating Conversational Web Services in Oracle Fusion Middleware Developing JAX-RPC Web Services for Oracle WebLogic Server for detailed information and examples of using this annotation.

### Attributes

**Table 3-13 Attributes of the weblogic.jws.Conversation JWS Annotation Tag**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Data Type</th>
<th>Required?</th>
</tr>
</thead>
</table>
| value | Specifies the phase of a conversation that the annotated method implements. Possible values are:  
• Phase.START  
  Specifies that the method starts a new conversation. A call to this method creates a new conversation ID and context, and resets its idle and age timer.  
• Phase.CONTINUE  
  Specifies that the method is part of a conversation in progress. A call to this method resets the idle timer. This method must always be called after the start method and before the finish method.  
• Phase.FINISH  
  Specifies that the method explicitly finishes a conversation in progress. Default value is Phase.CONTINUE | enum | No |

### Example

The following sample snippet shows a JWS file that contains three methods, start, middle, and finish) that are annotated with the @Conversation annotation to specify the start, continue, and finish phases, respectively, of a conversation.

```java
public class ConversationalServiceImpl implements Serializable {  
    @WebMethod
```
The following sections describe the annotation in detail.

**Description**

**Target:** Class

Specifies that a JWS file implements a conversational Web Service.

You are not required to use this annotation to specify that a Web Service is conversational; by simply annotating a single method with the `@Conversation` annotation, all the methods of the JWS file are automatically tagged as conversational. Use the class-level `@Conversational` annotation only if you want to change some of the conversational behavior or if you want to clearly show at the class level that the JWS is conversational.

If you do use the `@Conversational` annotation in your JWS file, you can specify it without any attributes if their default values suit your needs. However, if you want to change values such as the maximum amount of time that a conversation can remain idle, the maximum age of a conversation, and so on, specify the appropriate attribute.

See Creating Conversational Web Services in Oracle Fusion Middleware Developing JAX-RPC Web Services for Oracle WebLogic Server for detailed information and examples of using this annotation.
### Attributes

Table 3-14 Attributes of the weblogic.jws.Conversational JWS Annotation Tag

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Data Type</th>
<th>Required?</th>
</tr>
</thead>
</table>
| maxIdleTime | Specifies the amount of time that a conversation can remain idle before it is finished by WebLogic Server. Activity is defined by a client Web Service executing one of the phases of the conversation. Valid values are a number and one of the following terms:  
• seconds  
• minutes  
• hours  
• days  
• years  
For example, to specify a maximum idle time of ten minutes, specify the annotation as follows:  
@Conversational(maxIdleTime="10 minutes")  
If you specify a zero-length value (such as 0 seconds, or 0 minutes and so on), then the conversation never times out due to inactivity. Default value is 0 seconds. | String    | No         |
| maxAge    | The amount of time that a conversation can remain active before it is finished by WebLogic Server. Valid values are a number and one of the following terms:  
• seconds  
• minutes  
• hours  
• days  
• years  
For example, to specify a maximum age of three days, specify the annotation as follows:  
@Conversational(maxAge="3 days")  
Default value is 1 day. | String    | No         |
Table 3-14  (Cont.) Attributes of the weblogic.jws.Conversational JWS Annotation Tag

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Data Type</th>
<th>Required?</th>
</tr>
</thead>
</table>
| runAsStartUser     | Specifies whether the continue and finish phases of an existing conversation are run as the user who started the conversation.  
Typically, the same user executes the start, continue, and finish methods of a conversation, so that changing the value of this attribute has no effect. However, if you set the singlePrincipal attribute to false, which allows users different from the user who initiated the conversation to execute the continue and finish phases of an existing conversation, then the runAsStartUser attribute specifies which user the methods are actually "run as": the user who initiated the conversation or the different user who executes subsequent phases of the conversation.  
Valid values are true and false. Default value is false. | boolean   | No        |
| singlePrincipal    | Specifies whether users other than the one who started a conversation are allowed to execute the continue and finish phases of the conversation.  
Typically, the same user executes all phases of a conversation. However, if you set this attribute to false, then other users can obtain the conversation ID of an existing conversation and use it to execute later phases of the conversation.  
Valid values are true and false. Default value is false. | boolean   | No        |

Example

The following sample snippet shows how to specify that a JWS file implements a conversational Web Service. The maximum amount of time the conversation can be idle is ten minutes, and the maximum age of the conversation, regardless of activity, is one day. The continue and finish phases of the conversation can be executed by a user other than the one that started the conversation; if this happens, then the corresponding methods are run as the new user, not the original user.

```java
package examples.webservices.conversation;
...
@Conversational(maxIdleTime="10 minutes",
   maxAge="1 day",
   runAsStartUser=false,
   singlePrincipal=false )
public class ConversationalServiceImpl implements Serializable {
   ...
```
weblogic.jws.FileStore

The following sections describe the annotation in detail.

Description

**Target:** Class

Specifies that the Web Service does not use the default WebLogic Server default filestore to store internal state information, such as conversational state, but rather uses one specified by the programmer. If you do not specify this JWS annotation in your JWS file, the Web Service uses the default filestore configured for WebLogic Server.

You can also use this JWS annotation for reliable Web Services to store internal state.

If you deploy the Web Service in a cluster, be sure you specify the *logical name* of the filestore so that the same name of the filestore can be used on all servers in the cluster.

**Note:**

This annotation applies only to filestores, not to JDBC stores.

Attributes

**Table 3-15** Attributes of the weblogic.jws.FileStore JWS Annotation Tag

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Data Type</th>
<th>Required?</th>
</tr>
</thead>
<tbody>
<tr>
<td>storeName</td>
<td>Specifies the name of the filestore.</td>
<td>String</td>
<td>Yes</td>
</tr>
</tbody>
</table>

weblogic.jws.MessageBuffer

The following sections describe the annotation in detail.

Description

**Target:** Class, Method

Specifies which public methods of a JWS are buffered. If specified at the class-level, then all public methods are buffered; if you want only a subset of the methods to be buffered, specify the annotation at the appropriate method-level.

When a client Web Service invokes a buffered operation of a different WebLogic Web Service, WebLogic Server (hosting the invoked Web Service) puts the invoke message on a JMS queue and the actual invoke is dealt with later on when the WebLogic Server delivers the message from the top of the JMS queue to the Web Service implementation. The client does not need to wait for a response, but rather, continues on with its execution. For this reason, buffered operations (without any additional asynchronous features) can only return `void` and must be marked with the
@Oneway annotation. If you want to buffer an operation that returns a value, you must use asynchronous request-response from the invoking client Web Service. See Invoking a Web Service Using Asynchronous Request-Response in *Oracle Fusion Middleware Developing JAX-RPC Web Services for Oracle WebLogic Server* for more information.

Buffering works only between two Web Services in which one invokes the buffered operations of the other.

Use the optional attributes of @MessageBuffer to specify the number of times the JMS queue attempts to invoke the buffered Web Service operation until it is invoked successfully, and the amount of time between attempts.

Use the optional class-level @BufferQueue annotation to specify the JMS queue to which the invoke messages are queued. If you do not specify this annotation, the messages are queued to the default Web Service queue, weblogic.wsee.DefaultQueue.

See Creating Buffered Web Services in *Oracle Fusion Middleware Developing JAX-RPC Web Services for Oracle WebLogic Server* for detailed information and examples for using this annotation.

**Attributes**

**Table 3-16   Attributes of the weblogic.jws.MessageBuffer JWS Annotation Tag**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Data Type</th>
<th>Required?</th>
</tr>
</thead>
<tbody>
<tr>
<td>retryCount</td>
<td>Specifies the number of times that the JMS queue on the invoked WebLogic Server instance attempts to deliver the invoking message to the Web Service implementation until the operation is successfully invoked. Default value is 3.</td>
<td>int</td>
<td>No</td>
</tr>
</tbody>
</table>
| retryDelay   | Specifies the amount of time that elapses between message delivery retry attempts. The retry attempts are between the invoke message on the JMS queue and delivery of the message to the Web Service implementation. Valid values are a number and one of the following terms:
  * seconds
  * minutes
  * hours
  * days
  * years
For example, to specify a retry delay of two days, specify:
@MessageBuffer(retryDelay="2 days")

Default value is 5 seconds. | String    | No        |
Example

The following example shows a code snippet from a JWS file in which the public operation `sayHelloNoReturn` is buffered and the JMS queue to which WebLogic Server queues the operation invocation is called `my.buffer.queue`. The WebLogic Server instance that hosts the invoked Web Service tries a maximum of 10 times to deliver the invoke message from the JMS queue to the Web Service implementation, waiting 10 seconds between each retry. Only the relevant Java code is shown in the following snippet:

```java
package examples.webservices.buffered;
...
@WebService(name="BufferedPortType",
    serviceName="BufferedService",
    targetNamespace="http://example.org")
@BufferQueue(name="my.buffer.queue")
public class BufferedImpl {
...
    @WebMethod()
    @MessageBuffer(retryCount=10, retryDelay="10 seconds")
    @Oneway()
    public void sayHelloNoReturn(String message) {
        System.out.println("sayHelloNoReturn: " + message);
    }
}
```

weblogic.jws.Policies

The following sections describe the annotation in detail.

Description

**Target:** Class, Method

Specifies an array of `@weblogic.jws.Policy` annotations.

Use this annotation if you want to attach more than one WS-Policy files to a class or method of a JWS file. If you want to attach just one WS-Policy file, you can use the `@weblogic.jws.Policy` on its own.


This JWS annotation does not have any attributes.

Example

```java
@Policies(
    @Policy(uri="policy:firstPolicy.xml"),
    @Policy(uri="policy:secondPolicy.xml")
)
```
weblogic.jws.Policy

The following sections describe the annotation in detail.

Description

**Target:** Class, Method

Specifies that a WS-Policy file, which contains information about digital signatures, encryption, or Web Service reliable messaging, should be applied to the request or response SOAP messages.

This annotation can be used on its own to apply a single WS-Policy file to a class or method. If you want to apply more than one WS-Policy file to a class or method, use the `@weblogic.jws.Policies` annotation to group them together.

If this annotation is specified at the class level, the indicated WS-Policy file or files are applied to every public operation of the Web Service. If the annotation is specified at the method level, then only the corresponding operation will have the WS-Policy file applied.

By default, WS-Policy files are applied to both the request (inbound) and response (outbound) SOAP messages. You can change this default behavior with the `direction` attribute.

Also by default, the specified WS-Policy file is attached to the generated and published WSDL file of the Web Service so that consumers can view all the WS-Policy requirements of the Web Service. Use the `attachToWsdl` attribute to change this default behavior.

See Using Web Services Reliable Messaging in *Oracle Fusion Middleware Developing JAX-RPC Web Services for Oracle WebLogic Server* and Configuring Message-Level Security in *Securing WebLogic Web Services for Oracle WebLogic Server* for detailed information and examples of using this annotation.
### Attributes

#### Table 3-17 Attributes of the `weblogic.jws.Policy` JWS Annotation Tag

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Data Type</th>
<th>Required?</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>uri</strong></td>
<td>Specifies the location from which to retrieve the WS-Policy file. Use the <code>http:</code> prefix to specify the URL of a WS-Policy file on the Web. Use the <code>policy:</code> prefix to specify that the WS-Policy file is packaged in the Web Service archive file or in a shareable Java EE library of WebLogic Server, as shown in the following example: @Policy(uri=&quot;policy:MyPolicyFile.xml&quot;) If you are going to publish the WS-Policy file in the Web Service archive, the WS-Policy XML file must be located in either the <code>META-INF/policies</code> or <code>WEB-INF/policies</code> directory of the EJB JAR file (for EJB implemented Web Services) or WAR file (for Java class implemented Web Services), respectively. For information on publishing the WS-Policy file in a library, see Creating Shared Java EE Libraries and Optional Packages in Developing Applications for Oracle WebLogic Server.</td>
<td>String</td>
<td>Yes</td>
</tr>
<tr>
<td><strong>direction</strong></td>
<td>Specifies when to apply the policy: on the inbound request SOAP message, the outbound response SOAP message, or both (default). Valid values for this attribute are: • Policy.Direction.both • Policy.Direction.inbound • Policy.Direction.outbound The <strong>default value</strong> is <code>Policy.Direction.both</code>.</td>
<td>enum</td>
<td>No</td>
</tr>
<tr>
<td><strong>attachToWsdl</strong></td>
<td>Specifies whether the WS-Policy file should be attached to the WSDL that describes the Web Service. <strong>Valid values are</strong> <code>true</code> and <code>false</code>. <strong>Default value</strong> is <code>false</code>.</td>
<td>boolean</td>
<td>No</td>
</tr>
</tbody>
</table>

### Example

```java
@Policy(uri="policy:myPolicy.xml",
attachToWsdl=true,
direction=Policy.Direction.outbound)
```

**weblogic.jws.ReliabilityBuffer**

The following sections describe the annotation in detail.
Description

**Target:** Method

Specifies reliable messaging properties for an operation of a reliable Web Service, such as the number of times WebLogic Server should attempt to deliver the message from the JMS queue to the Web Service implementation, and the amount of time that the server should wait in between retries.

**Note:**

It is assumed when you specify this annotation in a JWS file that you have already enabled reliable messaging for the Web Service by also including a `@Policy` annotation that specifies a WS-Policy file that has Web Service reliable messaging policy assertions.

If you specify the `@ReliabilityBuffer` annotation, but do not enable reliable messaging with an associated WS-Policy file, then WebLogic Server ignores this annotation.

See Using Web Services Reliable Messaging in *Oracle Fusion Middleware Developing JAX-RPC Web Services for Oracle WebLogic Server* for detailed information about enabling Web Services reliable messaging for your Web Service.

Attributes

Table 3-18   Attributes of the weblogic.jws.ReliabilityBuffer JWS Annotation Tag

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Data Type</th>
<th>Required?</th>
</tr>
</thead>
<tbody>
<tr>
<td>retryCount</td>
<td>Specifies the number of times that the JMS queue on the destination WebLogic Server instance attempts to deliver the message from a client that invokes the reliable operation to the Web Service implementation. Default value is 3.</td>
<td>int</td>
<td>No</td>
</tr>
</tbody>
</table>
Table 3-18  (Cont.) Attributes of the weblogic.jws.ReliabilityBuffer JWS Annotation Tag

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Data Type</th>
<th>Required?</th>
</tr>
</thead>
<tbody>
<tr>
<td>retryDelay</td>
<td>Specifies the amount of time that elapses between message delivery retry attempts. The retry attempts are between the client's request message on the JMS queue and delivery of the message to the Web Service implementation. Valid values are a number and one of the following terms: • seconds • minutes • hours • days • years For example, to specify a retry delay of two days, specify: @ReliabilityBuffer(retryDelay=&quot;2 days&quot;) Default value is 5 seconds.</td>
<td>String</td>
<td>No</td>
</tr>
</tbody>
</table>

Example

The following sample snippet shows how to use the @ReliabilityBuffer annotation at the method-level to change the default retry count and delay of a reliable operation; only relevant Java code is shown:

```java
package examples.webservices.reliable;
import javax.jws.WebMethod;
import javax.jws.WebService;
import javax.jws.Oneway;
...
import weblogic.jws.ReliabilityBuffer;
import weblogic.jws.Policy;
@WebService(name="ReliableHelloWorldPortType",
            serviceName="ReliableHelloWorldService")
public class ReliableHelloWorldImpl {
    @WebMethod()
    @Oneway()
    @ReliabilityBuffer(retryCount=10, retryDelay="10 seconds")
    public void helloWorld(String input) {
        System.out.println(" Hello World "+ input);
    }
}
```
weblogic.jws.ReliabilityErrorHandler

The following sections describe the annotation in detail.

Description

**Target:** Method

Specifies the method that handles the error that results when a client Web Service invokes a reliable Web Service, but the client does not receive an acknowledgement that the reliable Web Service actually received the message.

This annotation is relevant only when you implement the Web Service reliable messaging feature; you specify the annotation in the client-side Web Service that invokes a reliable Web Service.

The method you annotate with the `@ReliabilityErrorHandler` annotation takes a single parameter of data type `weblogic.wsee.reliability.ReliabilityErrorContext`. You can use this context to get more information about the cause of the error, such as the operation that caused it, the target Web Service, the fault, and so on. The method must return `void`.

The single attribute of the `@ReliabilityErrorHandler` annotation specifies the variable into which you have previously injected the stub information of the reliable Web Service that the client Web Service is invoking; you inject this information in a variable using the `@weblogic.jws.ServiceClient` annotation.

Attributes

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Data Type</th>
<th>Required?</th>
</tr>
</thead>
<tbody>
<tr>
<td>target</td>
<td>Specifies the target stub name for which this method handles reliability failures.</td>
<td>String</td>
<td>Yes</td>
</tr>
</tbody>
</table>

Example

The following code snippet from a client Web Service that invokes a reliable Web Service shows how to use the `@ReliabilityErrorHandler` annotation; not all code is shown, and the code relevant to this annotation is shown in **bold**:

```java
package examples.webservices.reliable;
...
import weblogic.jws.ServiceClient;
import weblogic.jws.ReliabilityErrorHandler;
import examples.webservices.reliable.ReliableHelloWorldPortType;
import weblogic.wsee.reliability.ReliabilityErrorContext;
import weblogic.wsee.reliability.ReliableDeliveryException;
@WebService(name="ReliableClientPortType",
...
public class ReliableClientImpl
{
    @ServiceClient(
```
In the example, the `port` variable has been injected with the stub that corresponds to the `ReliableHelloWorldService` Web Service, and it is assumed that at some point in the client Web Service an operation of this stub is invoked. Because the `onReliableMessageDeliveryError` method is annotated with the `@ReliabilityErrorHandler` annotation and is linked with the `port` stub, the method is invoked if there is a failure in an invoke of the reliable Web Service. The reliable error handling method uses the `ReliabilityErrorContext` object to get more details about the cause of the failure.

weblogic.jws.ServiceClient

The following sections describe the annotation in detail.

**Description**

**Target:** Field

Specifies that the annotated variable in the JWS file is a stub used to invoke another WebLogic Web Service when using the following features:

- Web Service reliable messaging
- Asynchronous request-response
- Conversations

You use the reliable messaging and asynchronous request-response features only between two Web Services; this means, for example, that you can invoke a reliable Web Service operation only from within another Web Service, not from a stand-alone client. In the case of reliable messaging, the feature works between any two application servers that implement the WS-ReliableMessaging specification at [http://docs.oasis-open.org/ws-rx/wsrmm/200702/wsrmm-1.1-spec-os-01.pdf](http://docs.oasis-open.org/ws-rx/wsrmm/200702/wsrmm-1.1-spec-os-01.pdf). In the case of asynchronous request-response, the feature works only between two WebLogic Server instances.

You use the `@ServiceClient` annotation in the client Web Service to specify which variable is a port type for the Web Service described by the `@ServiceClient` attributes.
The Enterprise Application that contains the client Web Service must also include the stubs of the Web Service you are invoking; you generate the stubs with the clientgen Ant task.

See Oracle Fusion Middleware Developing JAX-RPC Web Services for Oracle WebLogic Server for additional information and examples of using the @ServiceClient annotation.

Attributes

Table 3-20 Attributes of the weblogic.jws.ServiceClient JWS Annotation Tag

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Data Type</th>
<th>Required?</th>
</tr>
</thead>
<tbody>
<tr>
<td>serviceName</td>
<td>Specifies the name of the Web Service that you are invoking. Corresponds to the name attribute of the &lt;service&gt; element in the WSDL of the invoked Web Service. If you used a JWS file to implement the invoked Web Service, this attribute corresponds to the serviceName attribute of the @WebService JWS annotation in the invoked Web Service.</td>
<td>String</td>
<td>Yes</td>
</tr>
<tr>
<td>portName</td>
<td>Specifies the name of the port of the Web Service you are invoking. Corresponds to the name attribute of the &lt;port&gt; child element of the &lt;service&gt; element. If you used a JWS file to implement the invoked Web Service, this attribute corresponds to the portName attribute of the @WLHttpTransport JWS annotation in the invoked Web Service. If you do not specify this attribute, it is assumed that the &lt;service&gt; element in the WSDL contains only one &lt;port&gt; child element, which @ServiceClient uses. If there is more than one port, the client Web Service returns a runtime exception.</td>
<td>String</td>
<td>No</td>
</tr>
<tr>
<td>wsdlLocation</td>
<td>Specifies the WSDL file that describes the Web Service you are invoking. If you do not specify this attribute, the client Web Service uses the WSDL file from which the clientgen Ant task created the Service implementation of the Web Service to be invoked.</td>
<td>String</td>
<td>No</td>
</tr>
<tr>
<td>endpointAddress</td>
<td>Specifies the endpoint address of the Web Service you are invoking. If you do not specify this attribute, the client Web Service uses the endpoint address specified in the WSDL file.</td>
<td>String</td>
<td>No</td>
</tr>
</tbody>
</table>

Example

The following JWS file excerpt shows how to use the @ServiceClient annotation in a client Web Service to annotate a field (port) with the stubs of the Web Service being invoked (called ReliableHelloWorldService whose WSDL is at the URL http://
package examples.webservices.reliable;
import javax.jws.WebService;
...
import weblogic.jws.ServiceClient;
import examples.webservices.reliable.ReliableHelloWorldPortType;
@WebService(...
public class ReliableClientImpl
{
    @ServiceClient(
        serviceName="ReliableHelloWorldService",
        portName="ReliableHelloWorldServicePort")
    private ReliableHelloWorldPortType port;
    @WebMethod
    public void callHelloWorld(String input, String serviceUrl)
    throws RemoteException {
        port.helloWorld(input);
        System.out.println(" Invoked the ReliableHelloWorld.helloWorld operation reliably.");
    }
}

weblogic.jws.StreamAttachments

The following sections describe the annotation in detail.

Description

**Target:** Class

Specifies that the WebLogic Web Services runtime use streaming APIs when reading the parameters of all methods of the Web Service. This increases the performance of Web Service operation invocation, in particular when the parameters are large, such as images.

You cannot use this annotation if you are also using the following features in the same Web Service:

- Conversations
- Reliable Messaging
- JMS Transport
- A proxy server between the client application and the Web Service it invokes

The `@StreamAttachments` annotation does not have any attributes.

Example

The following simple JWS file shows how to specify the `@StreamAttachments` annotation; the single method, `echoAttachment()`, simply takes a `DataHandler` parameter and echoes it back to the client application that invoked the Web Service operation. The WebLogic Web Services runtime uses streaming when reading the `DataHandler` content.
package examples.webservices.stream_attach;
import javax.jws.WebMethod;
import javax.jws.WebService;
import weblogic.jws.WLHttpTransport;
import weblogic.jws.StreamAttachments;
import javax.activation.DataHandler;
import java.rmi.RemoteException;
@WebService(name="StreamAttachPortType",
            serviceName="StreamAttachService",
            targetNamespace="http://example.org")
@WLHttpTransport(contextPath="stream_attach",
                   serviceUri="StreamAttachService",
                   portName="StreamAttachServicePort")
@StreamAttachments
/**
 * Example of stream attachments
 */
public class StreamAttachImpl {
    @WebMethod()
    public DataHandler echoAttachment(DataHandler dh) throws RemoteException {
        return dh;
    }
}

weblogic.jws.Transactional

The following sections describe the annotation in detail.

**Description**

**Target:** Class, Method

Specifies whether the annotated operation, or all the operations of the JWS file when
the annotation is specified at the class-level, runs or run inside of a transaction. By
default, the operations do not run inside of a transaction.

**Attributes**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Data Type</th>
<th>Required?</th>
</tr>
</thead>
<tbody>
<tr>
<td>value</td>
<td>Specifies whether the operation (when used at the method level) or all the operations of the Web Service (when specified at the class level) run inside of a transaction. Valid values are true and false. Default value is false.</td>
<td>boolean</td>
<td>No</td>
</tr>
<tr>
<td>timeout</td>
<td>Specifies a timeout value, in seconds, for the current transaction. The default value for this attribute is 30 seconds.</td>
<td>int</td>
<td>No</td>
</tr>
</tbody>
</table>
Example

The following example shows how to use the `@Transactional` annotation to specify that an operation of a Web Service executes as part of a transaction:

```java
package examples.webservices.transactional;
import javax.jws.WebMethod;
import javax.jws.WebService;
import weblogic.jws.wlHttpTransport;
import weblogic.jws.Transactional;
@WebService(name="TransactionPojoPortType",
        serviceName="TransactionPojoService",
        targetNamespace="http://example.org")
@WLHttpTransport(contextPath="transactionsPojo",
        serviceUri="TransactionPojoService",
        portName="TransactionPojoPort")
/**
 * This JWS file forms the basis of simple WebLogic Web Service with a single operation: sayHello. The operation executes as part of a transaction.
 */
public class TransactionPojoImpl {
    @WebMethod()
    @Transactional(value=true)
    public String sayHello(String message) {
        System.out.println("sayHello:" + message);
        return "Here is the message: " + message + "+";
    }
}
```

weblogic.jws.Types

The following sections describe the annotation in detail.

Description

**Target:** Method, Parameter

Specifies a comma-separated list of fully qualified Java class names of the alternative data types for a return type or parameter. The alternative data types must extend the data type specified in the method signature; if this is not the case, the `jwsc` Ant task returns a validation error when you compile the JWS file into a Web Service.

For example, assume you have created the `Address` base data type, and then created `USAAddress` and `CAAddress` that extend this base type. If the method signature specifies that it takes an `Address` parameter, you can annotate the parameter with the `@Types` annotation to specify that the public operation also takes `USAAddress` and `CAAddress` as a parameter, in addition to the base `Address` data type.

You can also use this annotation to restrict the data types that can be contained in parameters or return values of collection data types, such as `java.util.Collection` or `java.util.List`. By restricting the allowed contained data types, the generated WSDL is specific and unambiguous, and the Web Services runtime can do a better job of qualifying the parameters when a client application invokes a Web Service operation.
If you specify this annotation at the method-level, then it applies only to the return value. If you want the annotation to apply to parameters, you must specify it at the parameter-level for each relevant parameter.

**Attributes**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Data Type</th>
<th>Required?</th>
</tr>
</thead>
<tbody>
<tr>
<td>value</td>
<td>Comma-separated list of fully qualified class names for either the alternative data types that can also be used instead of the original data type, or the allowed data types contained in the collection-type parameter or return value.</td>
<td>String[]</td>
<td>Yes</td>
</tr>
</tbody>
</table>

**Example**

The following example shows a simple JWS file that uses the `@Types` annotation, with relevant Java code shown in **bold**:

```java
package examples.webservices.types;
import javax.jws.WebMethod;
import javax.jws.WebService;
import weblogic.jws.WLHttpTransport;
import weblogic.jws.Types;
import examples.webservices.types.BasicStruct;
@WebService(serviceName="TypesService",
            name="TypesPortType",
            targetNamespace="http://example.org")
@WLHttpTransport(contextPath="types",
                  serviceUri="TypesService",
                  portName="TypesServicePort")
public class TypesImpl {
    @WebMethod()
    @Types({"examples.webservices.types.ExtendedStruct"})
    public BasicStruct echoStruct(
        @Types({"examples.webservices.types.ExtendedStruct"}) BasicStruct struct)
    {
        System.out.println("echoStruct called");
        return struct;
    }
}
```

In the example, the signature of the `echoStruct()` method shows that it takes a `BasicStruct` value as both a parameter and a return value. However, because both the method and the `struct` parameter are annotated with the `@Types` annotation, a client application invoking the `echoStruct` operation can also pass it a parameter of data type `ExtendedStruct`; in this case the operation also returns an `ExtendedStruct` value. It is assumed that `ExtendedStruct` extends `BasicStruct`.

**weblogic.jws.WildcardBinding**

The following sections describe the annotation in detail.
Description

**Target:** Class

Specifies the XML Schema data type to which a wildcard class, such as `javax.xml.soap.SOAPElement` or `org.apache.xmlbeans.XmlObject`, binds. By default, these Java data types bind to the `<xsd:any>` XML Schema data type. By using this class-level annotation, you can specify that the wildcard classes bind to `<xsd:anyType>` instead.

Attributes

**Table 3-23** Attributes of the `weblogic.jws.WildcardBinding` JWS Annotation

<table>
<thead>
<tr>
<th>Tag</th>
<th>Name</th>
<th>Description</th>
<th>Data Type</th>
<th>Required?</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>className</td>
<td>Specifies the fully qualified name of the wildcard class for which this binding applies. Typical values are <code>javax.xml.soap.SOAPElement</code> and <code>org.apache.xmlbeans.XmlObject</code>.</td>
<td>String</td>
<td>Yes</td>
</tr>
</tbody>
</table>
|       | binding           | Specifies the XML Schema data type to which the wildcard class should bind. You can specify one of the following values:  
  * WildcardParticle.ANY  
  * WildcardParticle.ANYTYPE | enum      | Yes       |

Example

The following example shows how to use the `@WildcardBinding` annotation to specify that the Apache XMLBeans data type `XMLObject` should bind to the `<xsd:any>` XML Schema data type for this Web Service:

```java
@WildcardBindings({
  @WildcardBinding(className="org.apache.xmlbeans.XmlObject", binding=WildcardParticle.ANY),
  @WildcardBinding(className="org.apache.xmlbeans.XmlObject[]", binding=WildcardParticle.ANY)
})
public class SimpleImpl {
  ...
}
```

**weblogic.jws.WildcardBindings**

The following sections describe the annotation in detail.

Description

**Target:** Class

Specifies an array of `@weblogic.jws.WildcardBinding` annotations.

This JWS annotation does not have any attributes.

See `weblogic.jws.WildcardBinding` for an example.
weblogic.jws.WLHttpTransport

The following sections describe the annotation in detail.

Description

**Target:** Class

Specifies the context path and service URI sections of the URL used to invoke the Web Service over the HTTP transport, as well as the name of the port in the generated WSDL.

You can specify this annotation only once (maximum) in a JWS file.

Attributes

Table 3-24 Attributes of the weblogic.jws.WLHttpTransport JWS Annotation

<table>
<thead>
<tr>
<th>Tag</th>
<th>Description</th>
<th>Data Type</th>
<th>Required?</th>
</tr>
</thead>
<tbody>
<tr>
<td>contextPath</td>
<td>Context path of the Web Service. You use this value in the URL that invokes the Web Service. For example, assume you set the context path for a Web Service to financial; a possible URL for the WSDL of the deployed WebLogic Web Service is as follows: <a href="http://hostname:7001/financial/GetQuote?WSDL">http://hostname:7001/financial/GetQuote?WSDL</a>. The default value of this attribute is the name of the JWS file, without its extension. For example, if the name of the JWS file is HelloWorldImpl.java, then the default value of its contextPath is HelloWorldImpl.</td>
<td>String</td>
<td>No</td>
</tr>
<tr>
<td>serviceUri</td>
<td>Web Service URI portion of the URL. You use this value in the URL that invokes the Web Service. For example, assume you set this attribute to GetQuote; a possible URL for the deployed WSDL of the service is as follows: <a href="http://hostname:7001/financial/GetQuote?WSDL">http://hostname:7001/financial/GetQuote?WSDL</a>. The default value of this attribute is the name of the JWS file, without its extension. For example, if the name of the JWS file is HelloWorldImpl.java, then the default value of its serviceUri is HelloWorldImpl.</td>
<td>String</td>
<td>No</td>
</tr>
</tbody>
</table>
Table 3-24 (Cont.) Attributes of the weblogic.jws.WLHttpTransport JWS Annotation Tag

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Data Type</th>
<th>Required?</th>
</tr>
</thead>
<tbody>
<tr>
<td>portName</td>
<td>The name of the port in the generated WSDL. This attribute maps to the name attribute of the &lt;port&gt; element in the WSDL. The default value of this attribute is based on the @javax.jws.WebService annotation of the JWS file. In particular, the default portName is the value of the name attribute of @WebService annotation, plus the actual text SoapPort. For example, if @WebService.name is set to MyService, then the default portName is MyServiceSoapPort.</td>
<td>String</td>
<td>No</td>
</tr>
</tbody>
</table>

Example

```java
@WLHttpTransport(contextPath="complex",
  serviceUri="ComplexService",
  portName="ComplexServicePort")
```

weblogic.jws.WLHttpsTransport

The following sections describe the annotation in detail.

Description

**Target:** Class

---

**Note:**

The `@weblogic.jws.WLHttpsTransport` annotation is deprecated as of version 9.2 of WebLogic Server. You should use the `@weblogic.jws.WLHttpTransport` annotation instead because it now supports both the HTTP and HTTPS protocols. If you want client applications to access the Web Service using only the HTTPS protocol, then you must specify the `@weblogic.jws.security.UserDataConstraint` JWS annotation in your JWS file.

Specifies the context path and service URI sections of the URL used to invoke the Web Service over the HTTPS transport, as well as the name of the port in the generated WSDL.

You can specify this annotation only once (maximum) in a JWS file.
Attributes

Table 3-25    Attributes of the weblogic.jws.WLHttpsTransport JWS Annotation

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Data Type</th>
<th>Required?</th>
</tr>
</thead>
<tbody>
<tr>
<td>contextPath</td>
<td>Context path of the Web Service. You use this value in the URL that invokes the Web Service. For example, assume you set the context path for a Web Service to financial; a possible URL for the WSDL of the deployed WebLogic Web Service is as follows: <a href="https://hostname:7001/financial/GetQuote?WSDL">https://hostname:7001/financial/GetQuote?WSDL</a> The default value of this attribute is the name of the JWS file, without its extension. For example, if the name of the JWS file is HelloWorldImpl.java, then the default value of its contextPath is HelloWorldImpl.</td>
<td>String</td>
<td>No</td>
</tr>
<tr>
<td>serviceUri</td>
<td>Web Service URI portion of the URL. You use this value in the URL that invokes the Web Service. For example, assume you set this attribute to GetQuote; a possible URL for the deployed WSDL of the service is as follows: <a href="https://hostname:7001/financial/GetQuote?WSDL">https://hostname:7001/financial/GetQuote?WSDL</a> The default value of this attribute is the name of the JWS file, without its extension. For example, if the name of the JWS file is HelloWorldImpl.java, then the default value of its serviceUri is HelloWorldImpl.</td>
<td>String</td>
<td>No</td>
</tr>
<tr>
<td>portName</td>
<td>The name of the port in the generated WSDL. This attribute maps to the name attribute of the &lt;port&gt; element in the WSDL. The default value of this attribute is based on the @javax.jws.WebService annotation of the JWS file. In particular, the default portName is the value of the name attribute of @WebService annotation, plus the actual text SoapPort. For example, if @WebService.name is set to MyService, then the default portName is MyServiceSoapPort.</td>
<td>String</td>
<td>No</td>
</tr>
</tbody>
</table>

Example

@WLHttpsTransport(portName="helloSecurePort", contextPath="secure", serviceUri="SimpleSecureBean")

weblogic.jws.WLJmsTransport

The following sections describe the annotation in detail.
Description

**Target:** Class

Specifies the context path and service URI sections of the URL used to invoke the Web Service over the JMS transport, as well as the name of the port in the generated WSDL. You also use this annotation to specify the JMS queue to which WebLogic Server queues the SOAP request messages from invokes of the operations.

You can specify this annotation only once (maximum) in a JWS file.

Attributes

**Table 3-26  Attributes of the weblogic.jws.WLJmsTransport JWS Annotation**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Data Type</th>
<th>Required?</th>
</tr>
</thead>
<tbody>
<tr>
<td>contextPath</td>
<td>Context path (or context root) of the Web Service. You use this value in the URL that invokes the Web Service.</td>
<td>String</td>
<td>No</td>
</tr>
<tr>
<td>serviceUri</td>
<td>Web Service URI portion of the URL used by client applications to invoke the Web Service.</td>
<td>String</td>
<td>No</td>
</tr>
</tbody>
</table>
| queue             | The JNDI name of the JMS queue that you have configured for the JMS transport. See Using JMS Transport as the Connection Protocol in Oracle Fusion Middleware Developing JAX-RPC Web Services for Oracle WebLogic Server for details about using JMS transport.  
                   |                                                                                                                                             | String    | No        |
| portName          | The name of the port in the generated WSDL. This attribute maps to the `name` attribute of the `<port>` element in the WSDL.  
                   | If you do not specify this attribute, the `jwsc` generates a default name based on the name of the class that implements the Web Service.                                                              | String    | No        |
| connectionFactory | The JNDI name of the JMS connection factory that you have configured for the JMS transport. See Using JMS Transport as the Connection Protocol in Oracle Fusion Middleware Developing JAX-RPC Web Services for Oracle WebLogic Server for details about using JMS transport. | String    | Yes       |
Example

The following example shows how to specify that the JWS file implements a Web Service that is invoked using the JMS transport. The JMS queue to which WebLogic Server queues SOAP message requests from invokes of the service operations is JMSTransportQueue; it is assumed that this JMS queue has already been configured for WebLogic Server.

```java
WLJmsTransport(contextPath="transports",
               serviceUri="JMSTransport",
               queue="JMSTransportQueue",
               portName="JMSTransportServicePort")
```

weblogic.jws.WSDL

The following sections describe the annotation in detail.

Description

**Target:** Class

Specifies whether to expose the WSDL of a deployed WebLogic Web Service.

By default, the WSDL is exposed at the following URL:

```
http://[host]:[port]/[contextPath]/[serviceUri]?WSDL
```

where:

- *host* refers to the computer on which WebLogic Server is running.
- *port* refers to the port number on which WebLogic Server is listening (default value is 7001).
- *contextPath* and *serviceUri* refer to the value of the `contextPath` and `serviceUri` attributes, respectively, of the `@WLHttpTransport` JWS annotation of the JWS file that implements your Web Service.

For example, assume you used the following `@WLHttpTransport` annotation:

```java
@WLHttpTransport(portName="helloPort",
                  contextPath="hello",
                  serviceUri="SimpleImpl")
```

The URL to get view the WSDL of the Web Service, assuming the service is running on a host called ariel at the default port number, is:

```
```
Attributes

Table 3-27  Attributes of the weblogic.jws.WSDL JWS Annotation Tag

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Data Type</th>
<th>Required?</th>
</tr>
</thead>
<tbody>
<tr>
<td>exposed</td>
<td>Specifies whether to expose the WSDL of a deployed Web Service. Valid values are true and false. Default value is true, which means that by default the WSDL is exposed.</td>
<td>boolean</td>
<td>No</td>
</tr>
</tbody>
</table>

Example

The following use of the @WSDL annotation shows how to specify that the WSDL of a deployed Web Service not be exposed; only relevant Java code is shown:

```java
package examples.webservices;
import weblogic.jws.WSDL;
@WebService(name="WsdlAnnotationPortType",
            serviceName="WsdlAnnotationService",
            targetNamespace="http://example.org")
@WSDL(exposed=false)
public class WsdlAnnotationImpl {
    ...
}
```

weblogic.jws.security.CallbackRolesAllowed

The following sections describe the annotation in detail.

Description

**Target:** Method, Field

Specifies an array of @SecurityRole JWS annotations that list the roles that are allowed to invoke the callback methods of the Web Service. A user that is mapped to an unspecified role, or is not mapped to any role at all, would not be allowed to invoke the callback methods.

If you use this annotation at the field level, then the specified roles are allowed to invoke all callback operations of the Web Service. If you use this annotation at the method-level, then the specified roles are allowed to invoke only that callback method. If specified at both levels, the method value overrides the field value if there is a conflict.

Attributes

Table 3-28  Attributes of the weblogic.jws.security.CallbackRolesAllowed JWS Annotation Tag

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Data Type</th>
<th>Required?</th>
</tr>
</thead>
<tbody>
<tr>
<td>value</td>
<td>Array of @weblogic.jws.security.RolesAllowed that list the roles allowed to invoke the callback methods.</td>
<td>String[]</td>
<td>Yes</td>
</tr>
</tbody>
</table>
Example

The following example shows how to use the @CallbackRolesAllowed annotation at the method level to specify that the role engineer is allowed to invoke the callback method:

```java
@CallbackMethod(target="port", operation="callbackOperation")
@CallbackRolesAllowed(@SecurityRole(role="engineer", mapToPrincipals="shackell"))
public void callbackHandler(String msg) {
    System.out.println (msg);
}
```

we3blogic.jws.security.RolesAllowed

The following sections describe the annotation in detail.

Description

**Target:** Class, Method

Specifies whether to enable basic authentication for a Web Service. In particular, it specifies an array of @SecurityRole JWS annotations that describe the list of roles that are allowed to invoke the Web Service. A user that is mapped to an unspecified role, or is not mapped to any role at all, would not be allowed to invoke the Web Service.

If you use this annotation at the class-level, then the specified roles are allowed to invoke all operations of the Web Service. To specify roles for just a specific set of operations, specify the annotation at the operation-level.

Attributes

Table 3-29 Attributes of the weblogic.jws.security.RolesAllowed JWS Annotation Tag

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Data Type</th>
<th>Required?</th>
</tr>
</thead>
<tbody>
<tr>
<td>value</td>
<td>Array of @we3blogic.jws.security.RolesAllowed that list the roles allowed to invoke the Web Service methods.</td>
<td>String[]</td>
<td>Yes</td>
</tr>
</tbody>
</table>

Example

```java
package examples.webservices.security_roles;
...
import we3blogic.jws.security.RolesAllowed;
import we3blogic.jws.security.SecurityRole;
@WebService(name="SecurityRolesPortType",
            serviceName="SecurityRolesService",
            targetNamespace="http://example.org")
@RolesAllowed ( {  
    @SecurityRole (role="manager",
                    mapToPrincipals={ "juliet","amanda" }),
    @SecurityRole (role="vp")
})
```
public class SecurityRolesImpl {
...

In the example, only the roles manager and vp are allowed to invoke the Web Service. Within the context of the Web Service, the users juliet and amanda are assigned the role manager. The role vp, however, does not include a mapToPrincipals attribute, which implies that users have been mapped to this role externally. It is assumed that you have already added the two users (juliet and amanda) to the WebLogic Server security realm.

weblogic.jws.security.RolesReferenced

Description

Target: Class

Specifies the list of role names that reference actual roles that are allowed to invoke the Web Service. In particular, it specifies an array of @SecurityRoleRef JWS annotations, each of which describe a link between a referenced role name and an actual role defined by a @SecurityRole annotation.

This JWS annotation does not have any attributes.

Example

package examples.webservices.security_roles;
...
import weblogic.jws.security.RolesAllowed;
import weblogic.jws.security.SecurityRole;
import weblogic.jws.security.RolesReferenced;
import weblogic.jws.security.SecurityRoleRef;
@WebService(name="SecurityRolesPortType",
    serviceName="SecurityRolesService",
    targetNamespace="http://example.org")
@RolesAllowed (  {
    @SecurityRole (role="manager",
        mapToPrincipals={ "juliet","amanda "}),
    @SecurityRole (role="vp")
} )
@RolesReferenced (  
    @SecurityRoleRef (role="mgr", link="manager")
)
public class SecurityRolesImpl {
...

In the example, the role mgr is linked to the role manager, which is allowed to invoke the Web Service. This means that any user who is assigned to the role of mgr is also allowed to invoke the Web Service.

weblogic.jws.security.RunAs

The following sections describe the annotation in detail.
Description

**Target: Class**

Specifies the role and user identity which actually runs the Web Service in WebLogic Server.

For example, assume that the @RunAs annotation specifies the roleA role and userA principal. This means that even if the Web Service is invoked by userB (mapped to roleB), the relevant operation is actually executed internal as userA.

Attributes

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Data Type</th>
<th>Required?</th>
</tr>
</thead>
<tbody>
<tr>
<td>role</td>
<td>Specifies the role which the Web Service should be run as.</td>
<td>String</td>
<td>Yes</td>
</tr>
<tr>
<td>mapToPrincipal</td>
<td>Specifies the principal user that maps to the role. It is assumed that you have already configured the specified principal (user) as a valid WebLogic Server user, typically using the WebLogic Server Administration Console. See Create users in the Oracle WebLogic Server Administration Console Online Help for details.</td>
<td>String</td>
<td>Yes</td>
</tr>
</tbody>
</table>

Example

```java
package examples.webservices.security_roles;
import weblogic.jws.security.RunAs;
...
@WebService(name="SecurityRunAsPortType",
            serviceName="SecurityRunAsService",
            targetNamespace="http://example.org")
@RunAs (role="manager", mapToPrincipal="juliet")
public class SecurityRunAsImpl {
...
```

The example shows how to specify that the Web Service is always run as user juliet, mapped to the role manager, regardless of who actually invoked the Web Service.

**weblogic.jws.security.SecurityRole**

The following sections describe the annotation in detail.

Description

**Target: Class, Method**

Specifies the name of a role that is allowed to invoke the Web Service. This annotation is always specified in the JWS file as a member of a @RolesAllowed array.
When a client application invokes the secured Web Service, it specifies a user and password as part of its basic authentication. It is assumed that an administrator has already configured the user as a valid WebLogic Server user using the WebLogic Server Administration Console; for details see Create Users in the Oracle WebLogic Server Administration Console Online Help.

The user that is going to invoke the Web Service must also be mapped to the relevant role. You can perform this task in one of the following two ways:

- Use the WebLogic Server Administration Console to map the user to the role. In this case, you do not specify the mapToPrincipals attribute of the @SecurityRole annotation. See Add Users to Roles in the Oracle WebLogic Server Administration Console Online Help.
- Map the user to a role only within the context of the Web Service by using the mapToPrincipals attribute to specify one or more users.

To specify that multiple roles are allowed to invoke the Web Service, include multiple @SecurityRole annotations within the @RolesAllowed annotation.

### Attributes

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Data Type</th>
<th>Required?</th>
</tr>
</thead>
<tbody>
<tr>
<td>role</td>
<td>The name of the role that is allowed to invoke the Web Service.</td>
<td>String</td>
<td>Yes</td>
</tr>
<tr>
<td>mapToPrincipals</td>
<td>An array of user names that map to the role.</td>
<td>String[]</td>
<td>No</td>
</tr>
</tbody>
</table>

If you do not specify this attribute, it is assumed that you have externally defined the mapping between users and the role, typically using the WebLogic Server Administration Console.

### Example

```java
table
```

In the example, only the roles manager and vp are allowed to invoke the Web Service. Within the context of the Web Service, the users juliet and amanda are assigned the role manager. The role vp, however, does not include a mapToPrincipals attribute, which implies that users have been mapped to this role externally. It is assumed that
you have already added the two users (juliet and amanda) to the WebLogic Server security realm.

**weblogic.jws.security.SecurityRoleRef**

The following sections describe the annotation in detail.

**Description**

**Target:** Class

Specifies a role name reference that links to an already-specified role that is allowed to invoke the Web Service.

Users that are mapped to the role reference can invoke the Web Service as long as the referenced role is specified in the `@RolesAllowed` annotation of the Web Service.

**Attributes**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Data Type</th>
<th>Required?</th>
</tr>
</thead>
<tbody>
<tr>
<td>role</td>
<td>Name of the role reference.</td>
<td>String</td>
<td>Yes</td>
</tr>
<tr>
<td>link</td>
<td>Name of the already-specified role that is allowed to invoke the Web Service. The value of this attribute corresponds to the value of the role attribute of a <code>@SecurityRole</code> annotation specified in the same JWS file.</td>
<td>String</td>
<td>Yes</td>
</tr>
</tbody>
</table>

**Example**

```java
package examples.webservices.security_roles;
...
import weblogic.jws.security.RolesAllowed;
import weblogic.jws.security.SecurityRole;
import weblogic.jws.security.RolesReferenced;
import weblogic.jws.security.SecurityRoleRef;
@WebService(name="SecurityRolesPortType",
    serviceName="SecurityRolesService",
    targetNamespace="http://example.org")
@RolesAllowed ( { }
    @SecurityRole (role="manager",
        mapToPrincipals={ "juliet","amanda" } ),
    @SecurityRole (role="vp")
 )
@RolesReferenced ( 
    @SecurityRoleRef (role="mgr", link="manager"
)
)
public class SecurityRolesImpl {
...
```
In the example, the role \texttt{mgr} is linked to the role \texttt{manager}, which is allowed to invoke the Web Service. This means that any user who is assigned to the role of \texttt{mgr} is also allowed to invoke the Web Service.

\textbf{\texttt{weblogic.jws.security UserDataConstraint}}

The following sections describe the annotation in detail.

\textbf{Description}

\textbf{Target: Class}

Specifies whether the client is required to use the HTTPS transport when invoking the Web Service.

WebLogic Server establishes a Secure Sockets Layer (SSL) connection between the client and Web Service if the \texttt{transport} attribute of this annotation is set to either \texttt{Transport.INTEGRAL} or \texttt{Transport.CONFIDENTIAL} in the JWS file that implements the Web Service.

If you specify this annotation in your JWS file, you must also specify the \texttt{weblogic.jws.WLHttpTransport} annotation (or the \texttt{<WLHttpTransport>} element of the \texttt{jwsc Ant task}) to ensure that an HTTPS binding is generated in the WSDL file by the \texttt{jwsc Ant task}.

\textbf{Attributes}

\begin{table}[h]
\centering
\begin{tabular}{|l|p{12cm}|c|c|}
\hline
\textbf{Name} & \textbf{Description} & \textbf{Data Type} & \textbf{Required?} \\
\hline
\texttt{transport} & Specifies whether the client is required to use the HTTPS transport when invoking the Web Service. Valid values are:
\begin{itemize}
  \item \texttt{Transport.NONE}—Specifies that the Web Service does not require any transport guarantees.
  \item \texttt{Transport.INTEGRAL}—Specifies that the Web Service requires that the data be sent between the client and Web Service in such a way that it cannot be changed in transit.
  \item \texttt{Transport.CONFIDENTIAL}—Specifies that the Web Service requires that data be transmitted so as to prevent other entities from observing the contents of the transmission.
\end{itemize}
& \texttt{enum} & \texttt{No} \\
\hline
\end{tabular}
\caption{Attributes of the weblogic.jws.security UserDataConstraint JWS Annotation}
\end{table}

\textbf{Example}

\begin{verbatim}
package examples.webservices.security_https;
import weblogic.jws.security.UserDataConstraint;
...
@WebService(name="SecurityHttpsPortType",
    serviceName="SecurityHttpsService",
    ...
weblogic.jws.security.WssConfiguration

The following sections describe the annotation in detail.

Description

**Target:** Class

Specifies the name of the Web Service security configuration you want the Web Service to use. If you do not specify this annotation in your JWS file, the Web Service is associated with the default security configuration (called default_wss) if it exists in your domain.

The @WssConfiguration annotation only makes sense if your Web Service is configured for message-level security (encryption and digital signatures). The security configuration, associated to the Web Service using this annotation, 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 digital signatures, and so on.

WebLogic Web Services are not required to be associated with a security configuration; if the default behavior of the Web Services security runtime is adequate then no additional configuration is needed. If, however, a Web Service requires different behavior from the default (such as using an X.509 certificate for identity, rather than the default username/password token), then the Web Service must be associated with a security configuration.

Before you can successfully invoke a Web Service that specifies a security configuration, you must use the WebLogic Server Administration Console to create it. For details, see Create a Web Services security configuration in the Oracle WebLogic Server Administration Console Online Help. For general information about message-level security, see Configuring Message-Level Security in Securing WebLogic Web Services for Oracle WebLogic Server.

---

**Note:**

All WebLogic Web Services packaged in a single Web Application must be associated with the same security configuration when using the @WssConfiguration annotation. This means, for example, that if a @WssConfiguration annotation exists in all the JWS files that implement the Web Services contained in a given Web Application, then the value attribute of each @WssConfiguration must be the same.

To specify that more than one Web Service be contained in a single Web Application when using the jwsc Ant task to compile the JWS files into Web Services, group the corresponding <jws> elements under a single <module> element.
Attributes

Table 3-34 Attributes of the weblogic.jws.security.WssConfiguration JWS Annotation Tag

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Data Type</th>
<th>Required?</th>
</tr>
</thead>
<tbody>
<tr>
<td>value</td>
<td>Specifies the name of the Web Service security configuration that is associated with this Web Service. The default configuration is called default_wss. You must create the security configuration (even the default one) using the WebLogic Server Administration Console before you can successfully invoke the Web Service.</td>
<td>String</td>
<td>Yes</td>
</tr>
</tbody>
</table>

Example

The following example shows how to specify that a Web Service is associated with the my_security_configuration security configuration; only the relevant Java code is shown:

```java
package examples.webservices.wss_configuration;
import javax.jws.WebService;
...
import weblogic.jws.security.WssConfiguration;
@WebService(...
...
@WssConfiguration(value="my_security_configuration")
public class WssConfigurationImpl {
...
```

weblogic.jws.soap.SOAPBinding

The following sections describe the annotation in detail.

Description

**Target:** Method

Specifies the mapping of a Web Service operation onto the SOAP message protocol.

This annotation is analogous to `@javax.jws.soap.SOAPBinding` except that it applies to a method rather than the class. With this annotation you can specify, for example, that one Web Service operation uses RPC-encoded SOAP bindings and another operation in the same Web Service uses document-literal-wrapped SOAP bindings.

**Note:** Because `@weblogic.jws.soap.SOAPBinding` and `@javax.jws.soap.SOAPBinding` have the same class name, be careful which annotation you are referring to when using it in your JWS file.
### Attributes

**Table 3-35** Attributes of the `weblogic.jws.soap.SOAPBinding` JWS Annotation

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Data Type</th>
<th>Required?</th>
</tr>
</thead>
<tbody>
<tr>
<td>style</td>
<td>Specifies the message style of the request and response SOAP messages of the invoked annotated operation. Valid values are: • SOAPBinding.Style.RPC • SOAPBinding.Style.DOCUMENT. Default value is SOAPBinding.Style.DOCUMENT.</td>
<td>enum</td>
<td>No</td>
</tr>
<tr>
<td>use</td>
<td>Specifies the formatting style of the request and response SOAP messages of the invoked annotated operation. Valid values are: • SOAPBinding.Use.LITERAL • SOAPBinding.Use.ENCODED Default value is SOAPBinding.Use.LITERAL.</td>
<td>enum</td>
<td>No</td>
</tr>
<tr>
<td>parameterStyle</td>
<td>Determines whether method parameters represent the entire message body, or whether the parameters are elements wrapped inside a top-level element named after the operation. Valid values are: • SOAPBinding.ParameterStyle.BARE • SOAPBinding.ParameterStyle.WRAPPED Default value is SOAPBinding.ParameterStyle.WRAPPED</td>
<td>enum</td>
<td>No</td>
</tr>
</tbody>
</table>

**Example**

The following simple JWS file shows how to specify that, by default, the operations of the Web Service use document-literal-wrapped SOAP bindings; you specify this by using the `@javax.jws.soap.SOAPBinding` annotation at the class-level. The example then shows how to specify different SOAP bindings for individual methods by using the `@weblogic.jws.soap.SOAPBinding` annotation at the method-level. In particular, the `sayHelloDocLitBare()` method uses document-literal-bare SOAP bindings, and the `sayHelloRPCEncoded()` method uses RPC-encoded SOAP bindings.

```java
package examples.webservices.soap_binding_method;
import javax.jws.WebMethod;
import javax.jws.WebService;
import javax.jws.soap.SOAPBinding;
import weblogic.jws.WLHttpTransport;
```
/* Simple JWS example that shows how to specify soap bindings for a method. */

public class SoapBindingMethodImpl {

    @WebMethod()
    @weblogic.jws.soap.SOAPBinding(
        style=SOAPBinding.Style.DOCUMENT,
        use=SOAPBinding.Use.LITERAL,
        parameterStyle=SOAPBinding.ParameterStyle.BARE
    )
    public String sayHelloDocLitBare(String message) {
        System.out.println("sayHelloDocLitBare" + message);
        return "Here is the message: ©" + message + "©";
    }

    @WebMethod()
    @weblogic.jws.soap.SOAPBinding(
        style=SOAPBinding.Style.RPC,
        use=SOAPBinding.Use.ENCODED
    )
    public String sayHelloRPCEncoded (String message) {
        System.out.println("sayHelloRPCEncoded" + message);
        return "Here is the message: ©" + message + "©";
    }
}

weblogic.jws.security.SecurityRoles (deprecated)

The following sections describe the annotation in detail.

**Description**

**Target:** Class, Method

**Note:**

The @weblogic.security.jws.SecurityRoles JWS annotation is deprecated beginning in WebLogic Server 9.0.

Specifies the roles that are allowed to access the operations of the Web Service.

If you specify this annotation at the class level, then the specified roles apply to all public operations of the Web Service. You can also specify a list of roles at the method level if you want to associate different roles to different operations of the same Web Service.

@WebService(name="SoapBindingMethodPortType",
    serviceName="SoapBindingMethodService",
    targetNamespace="http://example.org")

@SOAPBinding(style=SOAPBinding.Style.DOCUMENT,
    use=SOAPBinding.Use.LITERAL,
    parameterStyle=SOAPBinding.ParameterStyle.WRAPPED)

@WLHttpTransport(contextPath="soap_binding_method",
    serviceUri="SoapBindingMethodService",
    portName="SoapBindingMethodServicePort")
Note:

The @SecurityRoles annotation is supported only within the context of an EJB-implemented Web Service. For this reason, you can specify this annotation only inside of a JWS file that explicitly implements javax.ejb.SessionBean. See Securing Enterprise JavaBeans (EJBs) in Developing Applications with the WebLogic Security Service for conceptual information about what it means to secure access to an EJB. See Should You Implement a Stateless Session EJB? in Developing JAX-WS Web Services for Oracle WebLogic Server for information about explicitly implementing an EJB in a JWS file.

Attributes

Table 3-36  Attributes of the weblogic.jws.security.SecurityRoles JWS Annotation

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Data Type</th>
<th>Required?</th>
</tr>
</thead>
<tbody>
<tr>
<td>rolesAllowed</td>
<td>Specifies the list of roles that are allowed to access the Web Service. This annotation is the equivalent of the &lt;method-permission&gt; element in the ejb-jar.xml deployment descriptor of the stateless session EJB that implements the Web Service.</td>
<td>Array of String</td>
<td>No</td>
</tr>
<tr>
<td>rolesReferenced</td>
<td>Specifies a list of roles referenced by the Web Service. The Web Service may access other resources using the credentials of the listed roles. This annotation is the equivalent of the &lt;security-role-ref&gt; element in the ejb-jar.xml deployment descriptor of the stateless session EJB that implements the Web Service.</td>
<td>Array of String</td>
<td>No</td>
</tr>
</tbody>
</table>

Example

The following example shows how to specify, at the class-level, that the Web Service can be invoked only by the Admin role; only relevant parts of the example are shown:

```java
package examples.webservices.security_roles;
import javax.ejb.SessionBean;
import javax.ejb.SessionContext;
import weblogic.ejbgen.Session;
import javax.jws.WebService;
...
import weblogic.jws.security.SecurityRoles;
@Session ejbName="SecurityRolesEJB")
@WebService(...
// Specifies the roles who can invoke the entire Web Service
@SecurityRoles(rolesAllowed="Admin")
public class SecurityRolesImpl implements SessionBean { ...
```
weblogic.jws.security.SecurityIdentity (deprecated)

The following sections describe the annotation in detail.

Description

**Target:** Class

![Note:]

The `@weblogic.security.jws.SecurityIdentity` JWS annotation is deprecated beginning in WebLogic Server 9.1.

Specifies the identity assumed by the Web Service when it is invoked.

Unless otherwise specified, a Web Service assumes the identity of the authenticated invoker. This annotation allows the developer to override this behavior so that the Web Service instead executes as a particular role. The role must map to a user or group in the WebLogic Server security realm.

![Note:]

The `@SecurityIdentity` annotation only makes sense within the context of an EJB-implemented Web Service. For this reason, you can specify this annotation only inside of a JWS file that explicitly implements `javax.ejb.SessionBean`. See Securing Enterprise JavaBeans (EJBs) in Developing Applications with the WebLogic Security Service for conceptual information about what it means to secure access to an EJB. See Should You Implement a Stateless Session EJB? in Developing JAX-WS Web Services for Oracle WebLogic Server for information about explicitly implementing an EJB in a JWS file.

Attributes

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Data Type</th>
<th>Required?</th>
</tr>
</thead>
<tbody>
<tr>
<td>value</td>
<td>Specifies the role which the Web Service assumes when it is invoked. The role must map to a user or group in the WebLogic Server security realm.</td>
<td>String</td>
<td>Yes</td>
</tr>
</tbody>
</table>

Example

The following example shows how to specify that the Web Service, when invoked, runs as the Admin role:
package examples.webservices.security_roles;
import javax.ejb.SessionBean;
import javax.ejb.SessionContext;
import weblogic.ejbgen.Session;
import javax.jws.WebService;
...
import weblogic.jws.security.SecurityIdentity;
@Session(ejbName="SecurityRolesEJB")
@WebService(...
    // Specifies that the Web Service runs as the Admin role
    @SecurityIdentity( value="Admin")
    public class SecurityRolesImpl implements SessionBean {
        ...

weblogic.wsee.jws.jaxws.owsm.Property

The following sections describe the annotation in detail.

Description

**Target: Class**

Specifies a policy configuration property override.

Use this annotation with the `weblogic.wsee.jws.jaxws.owsm.SecurityPolicy` annotation to override a configuration property when attaching a policy to a web service client.

**Note:**

This annotation can be used for web service clients only. It is not supported for web service (server-side) policy attachment.

See Attaching Policies to Java EE Web Services and Clients Using Annotations in *Securing Web Services and Managing Policies with Oracle Web Services Manager* for detailed information and examples of using this annotation.

This JWS annotation does not have any attributes.

**Example**

```java
@SecurityPolicy(uri="policy:oracle/wss10_message_protection_client_policy",
    properties = {
        @Property(name="keystore.recipient.alias", value="mykey")
    })
```

weblogic.wsee.jws.jaxws.owsm.SecurityPolicies

The following sections describe the annotation in detail.

Description

**Target: Class**
Specifies an array of `@weblogic.wsee.jws.jaxws.owsm.SecurityPolicies` annotations.

Use this annotation if you want to attach more than one OWSM security policy to the class of a JWS file. If you want to attach just one OWSM security policy, you can use the `@weblogic.wsee.jws.jaxws.owsm.SecurityPolicy` annotation.

See Attaching Policies to Java EE Web Services and Clients Using Annotations in Securing Web Services and Managing Policies with Oracle Web Services Manager for detailed information and examples of using this annotation.

This JWS annotation does not have any attributes.

Example

```java
@SecurityPolicies(
    @SecurityPolicy(uri="oracle/wss_saml20_token_over_sll_service_policy"),
    @SecurityPolicy(uri="oracle/binding_authorization_permitall_policy")
)
```

`weblogic.wsee.jws.jaxws.owsm.SecurityPolicy`

The following sections describe the annotation in detail.

**Description**

**Target:** Class

Attaches an OWSM security policy file to the web service or client.

This annotation can be used on its own to apply a single OWSM security policy to a class. If you want to attach more than one OWSM security policy to a class, use the `@weblogic.wsee.jws.jaxws.owsm.SecurityPolicies` annotation to group them together.

See Attaching Policies to Java EE Web Services and Clients Using Annotations in Securing Web Services and Managing Policies with Oracle Web Services Manager for detailed information and examples of using this annotation.
**Attributes**

### Table 3-38  Attributes of the weblogic.wsee.jws.jaxws.owsm.SecurityPolicy

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Data Type</th>
<th>Required?</th>
</tr>
</thead>
<tbody>
<tr>
<td>uri</td>
<td>Specifies the name of the OWSM security policy. Use the policy: prefix to specify that the OWSM policy is packaged in the OWSM policy repository, as shown in the following example: @SecurityPolicy(uri=&quot;policy:oracle/wss_saml20_token_over_ssl_service_policy&quot;) For more information about the OWSM repository, see Managing the OWSM Repository in Securing Web Services and Managing Policies with Oracle Web Services Manager.</td>
<td>String</td>
<td>Yes</td>
</tr>
<tr>
<td>properties</td>
<td><strong>Note:</strong> This attribute can be specified for web service clients only. This attribute is not supported for web service (server-side) policy attachment. Specifies policy configuration override information. You specify one or more configuration property values using the weblogic.wsee.jws.jaxws.owsm.Property annotation, as described in weblogic.wsee.jws.jaxws.owsm.Property.</td>
<td>String</td>
<td>No</td>
</tr>
<tr>
<td>enabled</td>
<td>Specifies whether the OWSM policy file is enabled. Valid values are true and false. Default value is true.</td>
<td>boolean</td>
<td>No</td>
</tr>
</tbody>
</table>

**Examples**

The following example shows how to attach the **wss_saml20_token_over_ssl_service_policy** to a web service.

```java
@SecurityPolicy(uri="policy:oracle/wss_saml20_token_over_ssl_service_policy", enabled=true)
```

The following example shows how to attach the **wss10_message_protection_client_policy** to a web service client and override the keystore.recipient.alias configuration property.

```java
@SecurityPolicy(uri="policy:oracle/wss10_message_protection_client_policy", properties = {
   @Property(name="keystore.recipient.alias", value="mykey")
}, enabled=true)
```

**weblogic.wsee.jws.jaxws.owsm.SecurityPolicies**

The following sections describe the annotation in detail.
Description

**Target:** Class, Method


Use this annotation if you want to attach more than one Oracle Web Services Manager (Oracle WSM) WS-Policy files to a class or method of a JWS file. If you want to attach just one Oracle WSM WS-Policy file, you can use the `weblogic.wsee.jws.jaxws.owsm.SecurityPolicy` on its own.

See Using Oracle Web Service Security Policies in *Securing WebLogic Web Services for Oracle WebLogic Server* for detailed information and examples of using this annotation.

This JWS annotation does not have any attributes.

Example

```java
@SecurityPolicies(
    @SecurityPolicy(uri="policy:firstPolicy.xml"),
    @SecurityPolicy(uri="policy:secondPolicy.xml")
)
```

`weblogic.wsee.jws.jaxws.owsm.SecurityPolicy`

The following sections describe the annotation in detail.
Note:

As is true for all JWS annotations, the `@SecurityPolicy` annotation cannot be overridden at runtime, which means that the Oracle WSM WS-Policy file you specify at buildtime using the annotation will always be associated with the Web Service. This means, for example, that although you can view the associated Oracle WSM WS-Policy file at runtime using the Administration Console, you cannot delete (unassociate) it. You can, however, associate additional Oracle WSM WS-Policy files using the console; see Configuring Oracle WSM Security Policies in Administration Console in the Securing WebLogic Web Services for Oracle WebLogic Server for detailed instructions.

Attribute

Table 3-39  Attribute of the `weblogic.jws.SecurityPolicy` JWS Annotation Tag

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Data Type</th>
<th>Required?</th>
</tr>
</thead>
<tbody>
<tr>
<td>uri</td>
<td>Specifies the location from which to retrieve the Oracle WSM WS-Policy file. Use the <code>http:</code> prefix to specify the URL of an Oracle WSM WS-Policy file on the Web. Use the <code>policy:</code> prefix to specify that the Oracle WSM WS-Policy file is packaged in the Web Service archive file or in a shareable Java EE library of WebLogic Server, as shown in the following example: @SecurityPolicy(uri=&quot;policy:oracle/wss10_username_token_with_message_protection_server_policy&quot;)</td>
<td>String</td>
<td>Yes</td>
</tr>
</tbody>
</table>

Example

```
@SecurityPolicy(uri="policy:oracle/wss10_username_token_with_message_protection_server_policy")
```

weblogic.wsee.wstx.wsat.Transactional

The following sections describe the annotation in detail.

Description

**Target:** Class, Method

Specifies whether the annotated class or method runs inside of a web service atomic transaction.

If you specify the `@Transactional` annotation at the web service class level, the settings apply to all two-way synchronous methods defined by the service endpoint.
interface. You can override the flow type value at the method level; however, the version must be consistent across the entire transaction.

WebLogic web services enable interoperability with other external transaction processing systems, such as WebSphere, JBoss, Microsoft .NET, and so on, through the support of the following specifications:

- WS-AtomicTransaction Version (WS-AT) 1.0, 1.1, and 1.2: http://docs.oasis-open.org/ws-tx/wstx-wsat-1.2-spec-cs-01/wstx-wsat-1.2-spec-cs-01.html
- WS-Coordination Version 1.0, 1.1, and 1.2: http://docs.oasis-open.org/ws-tx/wstx-wscoor-1.2-spec-cs-01/wstx-wscoor-1.2-spec-cs-01.html

Attributes

Table 3-40 Attribute of the weblogic.wsee.wstx.wsat.Transactional Annotation

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Data Type</th>
<th>Required?</th>
</tr>
</thead>
<tbody>
<tr>
<td>version</td>
<td>Version of the web services atomic transaction coordination context that is used for web services and clients. For clients, it specifies the version used for outbound messages only. The value specified must be consistent across the entire transaction. Valid values include WSAT10, WSAT11, WSAT12, and DEFAULT. The DEFAULT value for web services is all three versions (driven by the inbound request); the DEFAULT value for web service clients is WSAT10. For example: @Transactional(version=Transactional.Version.WSAT10)</td>
<td>String</td>
<td>No</td>
</tr>
<tr>
<td>value</td>
<td>Whether the web service atomic transaction coordination context is passed with the transaction flow. For valid values, see Table 3-41.</td>
<td>String</td>
<td>No</td>
</tr>
</tbody>
</table>

The following table summarizes the valid values for flow type and their meaning on the web service and client. The table also summarizes the valid value combinations when configuring web service atomic transactions for an EJB-style web service that uses the @TransactionAttribute annotation.

Table 3-41 Flow Types Values

<table>
<thead>
<tr>
<th>Value</th>
<th>Web Service Client</th>
<th>Web Service</th>
<th>Valid EJB @TransactionAttribute Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>NEVER</td>
<td>Do not export transaction coordination context.</td>
<td>Do not import transaction coordination context.</td>
<td>NEVER, NOT_SUPPORTED, REQUIRED, REQUIRES_NEW, SUPPORTS</td>
</tr>
<tr>
<td>SUPPORTS (Default)</td>
<td>Export transaction coordination context if transaction is available.</td>
<td>Import transaction coordination context if available in the message.</td>
<td>REQUIRED, SUPPORTS</td>
</tr>
</tbody>
</table>
### Table 3-41 (Cont.) Flow Types Values

<table>
<thead>
<tr>
<th>Value</th>
<th>Web Service Client</th>
<th>Web Service</th>
<th>Valid EJB @TransactionAttribute Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>MANDATORY</td>
<td>Export transaction coordination context. An exception is thrown if there is no active transaction.</td>
<td>Import transaction coordination context. An exception is thrown if there is no active transaction.</td>
<td>MANDATORY, REQUIRED, SUPPORTS</td>
</tr>
</tbody>
</table>

### Example

```java
@Transactional(value = Transactional.TransactionFlowType.SUPPORTS,
version="Transactional.Versino.WSAT12````
Web Service Reliable Messaging Policy Assertion Reference

Oracle WebLogic Server supports the use of WS-Policy files to configure reliable message capabilities of a WebLogic web service that are running on a destination endpoint.

- **Overview of a WS-Policy File That Contains Web Service Reliable Messaging Assertions**
- **WS-Policy File With Web Service Reliable Messaging Assertions—Version 1.2 and 1.1**
- **WS-Policy File With Web Service Reliable Messaging Assertions—Version 1.0 (Deprecated)**

### Overview of a WS-Policy File That Contains Web Service Reliable Messaging Assertions

Use the @Policy JWS annotations in the JWS file that implements the web service to specify the name of the WS-Policy file that is associated with a web service. A WS-Policy file is an XML file that conforms to the WS-Policy specification at [http://www.w3.org/TR/ws-policy/](http://www.w3.org/TR/ws-policy/).

The root element of a WS-Policy file is always `<wsp:Policy>`. To configure web service reliable messaging, you first add a `<wsrmp:RMAssertion>` child element; its main purpose is to group all the reliable messaging policy assertions together. Then, you add child elements to `<wsrmp:RMAssertion>` to define the web service reliable messaging. All these assertions conform to the WS-PolicyAssertions specification.

WebLogic Server includes default WS-Policy files that contain typical reliable messaging assertions that you can use if you do not want to create your own WS-Policy file. The default WS-Policy files are defined in:

- **JAX-WS**: Pre-Packaged WS-Policy Files for Web Services Reliable Messaging and MakeConnection in *Developing JAX-WS Web Services for Oracle WebLogic Server*
- **JAX-RPC**: Pre-Packaged WS-Policy Files for Reliable Messaging in *Oracle Fusion Middleware Developing JAX-RPC Web Services for Oracle WebLogic Server*

For task-oriented information about creating a reliable WebLogic web service, see:

- **JAX-WS**: Using Web Services Reliable Messaging in *Developing JAX-WS Web Services for Oracle WebLogic Server*
- **JAX-RPC**: Using Web Services Reliable Messaging in *Oracle Fusion Middleware Developing JAX-RPC Web Services for Oracle WebLogic Server*
WS-Policy File With Web Service Reliable Messaging Assertions—Version 1.2 and 1.1

You can create a WS-Policy file with web service reliable messaging assertions that are based on Version 1.2 and 1.1 of the WS Reliable Messaging Policy Assertion (WS-RM Policy) namespace. A description of Version 1.2 of the WS-RM Policy namespace is available at http://docs.oasis-open.org/ws-rx/wsrmp/200702.

- Example of a WS-Policy File With Web Service Reliable Messaging Assertions 1.2 and 1.1
- Element Descriptions

Example of a WS-Policy File With Web Service Reliable Messaging Assertions 1.2 and 1.1

The following example shows a simple WS-Policy file used to configure reliable messaging for a WebLogic web service.

```xml
<?xml version="1.0"?>
    <wsp:Policy>
        <wsp:DeliveryAssurance>
            <wsp:ExactlyOnce/>
        </wsp:DeliveryAssurance>
    </wsp:Policy>
</wsp:Policy>
```

Element Descriptions

The element hierarchy of web service reliable messaging policy assertions in a WS-Policy file is shown below. Each element is described in more detail in the following sections.

Note:

You must enter the assertions in the ordered listed below.
wsp:Policy

Groups nested policy assertions.

wsrmp:DeliveryAssurance

Specifies the delivery assurance (or quality of service) of the web service. You can set one of the delivery assurances defined in the following table. If not set, the delivery assurance defaults to ExactlyOnce.

**Table 4-1 Delivery Assurances for Reliable Messaging**

<table>
<thead>
<tr>
<th>Delivery Assurance</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>wsrmp:AtMostOnce</td>
<td>Messages are delivered at most once, without duplication. It is possible that some messages may not be delivered at all.</td>
</tr>
<tr>
<td>wsrmp:AtLeastOnce</td>
<td>Every message is delivered at least once. It is possible that some messages are delivered more than once.</td>
</tr>
<tr>
<td>wsrmp:ExactlyOnce</td>
<td>Every message is delivered exactly once, without duplication. This value is enabled by default.</td>
</tr>
<tr>
<td>wsrmp:InOrder</td>
<td>Messages are delivered in the order that they were sent. This delivery assurance can be combined with one of the preceding three assurances. This value is enabled by default.</td>
</tr>
</tbody>
</table>

The delivery assurance must be enclosed by wsp:Policy element. For example:

```xml
<wsrmp:DeliveryAssurance>
  <wsp:Policy>
    <wsrmp:ExactlyOnce/>
  </wsp:Policy>
</wsrmp:DeliveryAssurance>
```

**wsrmp:RMAssertion**

Main web service reliable messaging assertion that groups all the other assertions under a single element. The presence of this assertion in a WS-Policy file indicates that the corresponding web service must be invoked reliably.

The following table summarizes the attributes of the wsrmp:RMAssertion element.

**Table 4-2 Attributes of <wsrmp:RMAssertion>**

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
<th>Required?</th>
</tr>
</thead>
<tbody>
<tr>
<td>optional</td>
<td>Specifies whether the web service requires the operations to be invoked reliably. Valid values for this attribute are true and false. Default value is false.</td>
<td>No</td>
</tr>
</tbody>
</table>

**wsrmp:SequenceSTR**

Specifies that in order to secure messages in a reliable sequence, the runtime will use the wsse:SecurityTokenReference that is referenced in the CreateSequence
message. You can only specify one security assertion; that is, you can specify `wsrmp:SequenceSTR` or `wsrmp:SequenceTransportSecurity`, but not both.

**wsrmp:SequenceTransportSecurity**

Specifies that in order to secure messages in a reliable sequence, the runtime will use the SSL transport session that is used to send the `CreateSequence` message. This assertion must be used in conjunction with the `sp:TransportBinding` assertion that requires the use of some transport-level security mechanism (for example, `sp:HttpsToken`). You can only specify one security assertion; that is, you can specify `wsrmp:SequenceSTR` or `wsrmp:SequenceTransportSecurity`, but not both.

## WS-Policy File With Web Service Reliable Messaging Assertions—Version 1.0 (Deprecated)

Oracle WebLogic Server supports the ability to create a WS-Policy file with web service reliable messaging assertions that are based on WS Reliable Messaging Policy Assertion 1.0. This specification is available at [http://schemas.xmlsoap.org/ws/2005/02/rm/policy/](http://schemas.xmlsoap.org/ws/2005/02/rm/policy/).

- Example of a WS-Policy File With Web Service Reliable Messaging Assertions
- Element Description

### Example of a WS-Policy File With Web Service Reliable Messaging Assertions

The following example shows a simple WS-Policy file used to configure reliable messaging for a WebLogic web service:

```xml
<?xml version="1.0"?>
<wsp:Policy
 xmlns:wsrm="http://schemas.xmlsoap.org/ws/2005/02/rm/policy"
 xmlns:beapolicy="http://www.bea.com/wsrm/policy"
>
 <wsrm:RMAssertion>
   <wsrm:InactivityTimeout Milliseconds="600000" />
   <wsrm:BaseRetransmissionInterval Milliseconds="3000" />
   <wsrm:ExponentialBackoff />
   <wsrm:AcknowledgementInterval Milliseconds="200" />
   <beapolicy:Expires Expires="P1D" optional="true"/>
 </wsrm:RMAssertion>
</wsp:Policy>
```

## Element Description

The element hierarchy of web service reliable messaging policy assertions in a WS-Policy file is shown below. Each element is described in more detail in the following sections.
You must enter the assertions in the order listed below.

wsp:Policy
  wsrm:RMAssertion
    wsrm:InactivityTimeout
    wsrm:BaseRetransmissionInterval
    wsrm:ExponentialBackoff
    wsrm:AcknowledgementInterval
  beapolicy:Expires
  beapolicy:QOS

beapolicy:Expires

Specifies an amount of time after which the reliable web service expires and does not accept any new sequences. Client applications invoking this instance of the reliable web service will receive an error if they try to invoke an operation after the expiration duration.

The default value of this element, if not specified in the WS-Policy file, is for the web service to never expires.

Table 4-3 Attributes of <beapolicy:Expires>

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
<th>Required?</th>
</tr>
</thead>
<tbody>
<tr>
<td>Expires</td>
<td>The amount of time after which the reliable web service expires. The format of this attribute conforms to the XML Schema duration at <a href="http://www.w3.org/TR/2001/REC-xmlschema-2-20010502/#duration">http://www.w3.org/TR/2001/REC-xmlschema-2-20010502/#duration</a> data type. For example, to specify that the reliable web service expires after 3 hours, specify Expires=&quot;P3H&quot;.</td>
<td>Yes</td>
</tr>
</tbody>
</table>

beapolicy:QOS

Specifies the delivery assurance (or Quality Of Service) of the web service:
Table 4-4  Attributes of <beapolicy:QOS>

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
<th>Required?</th>
</tr>
</thead>
</table>
| QOS       | Specifies the delivery assurance. You can specify exactly one of the following values:  
- AtMostOnce—Messages are delivered at most once, without duplication. It is possible that some messages may not be delivered at all.  
- AtLeastOnce—Every message is delivered at least once. It is possible that some messages be delivered more than once.  
- ExactlyOnce—Every message is delivered exactly once, without duplication. You can also add the InOrder string to specify that the messages be delivered in order.  
If you specify one of the XXXOnce values, but do not specify InOrder, then the messages are not guaranteed to be in order. This is different from the default value if the entire QOS element is not specified (exactly once in order).  
This attribute defaults to ExactlyOnce InOrder.  
Example: `<beapolicy:QOS QOS="AtMostOnce InOrder" />` | Yes |

wsrm:AcknowledgementInterval

Specifies the maximum interval, in milliseconds, in which the destination endpoint must transmit a stand alone acknowledgement.

A destination endpoint can send an acknowledgement on the return message immediately after it has received a message from a source endpoint, or it can send one separately in a stand alone acknowledgement. In the case that a return message is not available to send an acknowledgement, a destination endpoint may wait for up to the acknowledgement interval before sending a stand alone acknowledgement. If there are no unacknowledged messages, the destination endpoint may choose not to send an acknowledgement.

This assertion does not alter the formulation of messages or acknowledgements as transmitted. Its purpose is to communicate the timing of acknowledgements so that the source endpoint may tune appropriately.

This element is optional. If you do not specify this element, the default value is set by the store and forward (SAF) agent configured for the destination endpoint.

Table 4-5  Attributes of <wsrm:AcknowledgementInterval>

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
<th>Required?</th>
</tr>
</thead>
<tbody>
<tr>
<td>Milliseconds</td>
<td>Specifies the maximum interval, in milliseconds, in which the destination endpoint must transmit a stand alone acknowledgement.</td>
<td>Yes</td>
</tr>
</tbody>
</table>
wsrm:BaseRetransmissionInterval

Specifies the interval, in milliseconds, that the source endpoint waits after transmitting a message and before it retransmits the message.

If the source endpoint does not receive an acknowledgement for a given message within the interval specified by this element, the source endpoint retransmits the message. The source endpoint can modify this retransmission interval at any point during the lifetime of the sequence of messages. This assertion does not alter the formulation of messages as transmitted, only the timing of their transmission.

This element can be used in conjunctions with the <wsrm:ExponentialBackoff> element to specify that the retransmission interval will be adjusted using the algorithm specified by the <wsrm:ExponentialBackoff> element.

This element is optional. If you do not specify this element, the default value is set by the store and forward (SAF) agent configured for the source endpoint. If using the WebLogic Server Administration Console to configure the SAF agent, this value is labeled Retry Delay Base.

Table 4-6   Attributes of <wsrm:BaseRetransmissionInterval>

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
<th>Required?</th>
</tr>
</thead>
<tbody>
<tr>
<td>Milliseconds</td>
<td>Number of milliseconds the source endpoint waits to retransmit message.</td>
<td>Yes</td>
</tr>
</tbody>
</table>

wsrm:ExponentialBackoff

Specifies that the retransmission interval will be adjusted using the exponential backoff algorithm.

This element is used in conjunction with the <wsrm:BaseRetransmissionInterval> element. If a destination endpoint does not acknowledge a sequence of messages for the amount of time specified by <wsrm:BaseRetransmissionInterval>, the exponential backoff algorithm will be used for timing of successive retransmissions by the source endpoint, should the message continue to go unacknowledged.

The exponential backoff algorithm specifies that successive retransmission intervals should increase exponentially, based on the base retransmission interval. For example, if the base retransmission interval is 2 seconds, and the exponential backoff element is set in the WS-Policy file, successive retransmission intervals if messages continue to be unacknowledged are 2, 4, 8, 16, 32, and so on.

This element is optional. If not set, the same retransmission interval is used in successive retries, rather than the interval increasing exponentially.

This element has no attributes.

wsrm:InactivityTimeout

Specifies (in milliseconds) a period of inactivity for a sequence of messages. A sequence of messages is defined as a set of messages, identified by a unique sequence number, for which a particular delivery assurance applies; typically a sequence originates from a single source endpoint. If, during the duration specified by
this element, a destination endpoint has received no messages from the source endpoint, the destination endpoint may consider the sequence to have been terminated due to inactivity. The same applies to the source endpoint.

This element is optional. If it is not set in the WS-Policy file, then sequences never time-out due to inactivity.

Table 4-7 Attributes of <wsrm:InactivityTimeout>

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
<th>Required?</th>
</tr>
</thead>
<tbody>
<tr>
<td>Milliseconds</td>
<td>The number of milliseconds that defines a period of inactivity.</td>
<td>Yes</td>
</tr>
</tbody>
</table>

wsrm:RMAssertion

Main web service reliable messaging assertion that groups all the other assertions under a single element.

The presence of this assertion in a WS-Policy file indicates that the corresponding web service must be invoked reliably.

Table 4-8 Attributes of <wsrm:RMAssertion>

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
<th>Required?</th>
</tr>
</thead>
<tbody>
<tr>
<td>optional</td>
<td>Specifies whether the web service requires the operations to be invoked reliably.</td>
<td>No</td>
</tr>
<tr>
<td></td>
<td>Valid values for this attribute are true and false. Default value is false.</td>
<td></td>
</tr>
</tbody>
</table>
You use WS-Policy files to enable and configure MakeConnection on a web service. Use the @Policy JWS annotations in the JWS file that implements the web service to specify the name of the WS-Policy file that is associated with a web service. A WS-Policy file is an XML file that conforms to the WS-Policy specification at http://www.w3.org/TR/ws-policy/.

- Overview of a WS-Policy File That Contains MakeConnection Assertions
- Example of a WS-Policy File With MakeConnection and WS-Policy 1.5
- Element Descriptions

Note:
This section applies only to JAX-WS web services, and not to JAX-RPC web services.

Overview of a WS-Policy File That Contains MakeConnection Assertions

The root element of a WS-Policy file is always <wsp:Policy>. To configure web service MakeConnection, you simply add a <wsmc:MCSupported> child element. The policy assertions conform to the WS-PolicyAssertions specification.

WebLogic Server includes default WS-Policy files that contain typical MakeConnection assertions that you can use if you do not want to create your own WS-Policy file. The default WS-Policy files are defined in Pre-Packaged WS-Policy Files for Web Services Reliable Messaging and MakeConnection in Developing JAX-WS Web Services for Oracle WebLogic Server.

For task-oriented information about enabling and configuring MakeConnection, see Using Asynchronous Web Service Clients Through a Firewall (MakeConnection) in Developing JAX-WS Web Services for Oracle WebLogic Server.

The following sections describe how to create a WS-Policy file with web service MakeConnection assertions that are based on WS-MakeConnection specification at http://docs.oasis-open.org/ws-rx/wsmc/200702.
Example of a WS-Policy File With MakeConnection and WS-Policy 1.5

Learn how to create a simple WS-Policy file, based on WS-Policy 1.5, to configure MakeConnection for a WebLogic web service.

```xml
<?xml version="1.0"?>
<wsp15:Policy xmlns:wsp15="http://www.w3.org/ns/ws-policy"
    xmlns:wsmc="http://docs.oasis-open.org/ws-rx/wsmc/200702">
    <wsmc:MCSupported wsp15:Optional="true"/>
</wsp15:Policy>
```

Element Descriptions

The web service MakeConnection policy assertions in a WS-Policy file contain elements that are arranged in a specific hierarchy. Each element in that hierarchy, shown below, is described in more detail in the sections that follow.

```xml
wsp:Policy
    wsmc:MCSupported
```

**wsp:Policy**

Groups nested policy assertions.

**wsmc:MCSupported**

The presence of this assertion in a WS-Policy file indicates that the corresponding web service uses MakeConnection as the transport model.

The following table summarizes the attributes of the `wsmc:MCSupport` element.

**Table 5-1 Attributes of `<wsmc:MCSupport>`**

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
<th>Required?</th>
</tr>
</thead>
<tbody>
<tr>
<td>optional</td>
<td>Specifies whether MakeConnection must be used by the web service client. Valid values for this attribute are <code>true</code> and <code>false</code>. Default value is <code>true</code>. If set to <code>false</code>, both <code>ReplyTo</code> and <code>FaultTo</code> headers must contain MakeConnection anonymous URIs.</td>
<td>No</td>
</tr>
</tbody>
</table>
Oracle Web Services Security Policy Assertion Reference

Oracle WebLogic Server supports the ability to configure security assertions in a WebLogic web services security policy file that conforms to the OASIS WS-SecurityPolicy 1.2 specification. This specification is available at http://www.oasis-open.org/committees/download.php/21401/ws-securitpolicy-1.2-spec-cd-01.pdf.

Previous releases of WebLogic Server, released before the formulation of the OASIS WS-SecurityPolicy specification, used security policy files written under the WS-Policy specification, using a proprietary schema for web services security policy. WebLogic Server still supports the proprietary web services security policy files first included in WebLogic Server version 9.0, but this legacy policy format is deprecated and should not be used for new applications.

- Overview of a Policy File That Contains Security Assertions
- Example of a Policy File With Security Elements
- Element Description
- Using MessageParts To Specify Parts of the SOAP Messages that Must Be Encrypted or Signed

Note:

This section applies only to JAX-RPC web services using policies written under the Oracle web services security policy schema, and not to JAX-WS web services or to policies written under the OASIS WS-SecurityPolicy 1.2 specification.

Overview of a Policy File That Contains Security Assertions

You can use policy files to configure the message-level security of a WebLogic web service. Use the @Policy and @Policies JWS annotations in the JWS file that implements the web service to specify the name of the security policy file that is associated with a WebLogic web service.

A security policy file is an XML file that conforms to the WS-Policy specification at http://www-106.ibm.com/developerworks/library/specification/ws-polfram/. The root element of a WS-Policy file is always <wsp:Policy>. To configure message-level security, you add policy assertions that specify the type of tokens supported for authentication and how the SOAP messages should be encrypted and digitally signed.
Note:

These security policy assertions are based on the assertions described in the December 18, 2002 version of the Web Services Security Policy Language (WS-SecurityPolicy) specification. This means that although the exact syntax and usage of the assertions in WebLogic Server are different, they are similar in meaning to those described in the specification. The assertions are not based on the latest update of the specification (13 July 2005.)

Policy files using the Oracle web services security policy schema have the following namespace

```xml
<wsp:Policy
 xmlns:wssp="http://www.bea.com/wls90/security/policy"
 xmlns:wsu="http://docs.oasis-open.org/wss/2004/01/oasis-2004-01-wss-wssecurity-utility-1.0.xsd"
 xmlns:wls="http://www.bea.com/wls90/security/policy/wsee#part"
>
<wssp:Identity>
  <wssp:SupportedTokens>
    <wssp:SecurityToken TokenType="http://docs.oasis-open.org/wss/2004/01/oasis-2004-01-saml-token-profile-1.0#SAMLAssertionID">
      <wssp:Claims>
        <wssp:ConfirmationMethod>sender-vouches</wssp:ConfirmationMethod>
      </wssp:Claims>
    </wssp:SecurityToken>
  </wssp:SupportedTokens>
</wssp:Identity>
<wssp:Confidentiality>
  <wssp:KeyWrappingAlgorithm URI="http://www.w3.org/2001/04/xmlenc#rsa-1_5"/>
</wssp:Confidentiality>
</wsp:Policy>
```

This release of WebLogic Server also includes a large number of packaged policy files that conform to the OASIS WS-SecurityPolicy 1.2 specification. WS-SecurityPolicy 1.2 policy files and Oracle proprietary web services security policy schema files are not mutually compatible; you cannot use both types of policy file in the same web services security configuration. For information about using WS-SecurityPolicy 1.2 security policy files, see Using WS-SecurityPolicy 1.2 Policy Files in Securing WebLogic Web Services for Oracle WebLogic Server.


Example of a Policy File With Security Elements

Learn about the security elements that are contained within a security policy file.

```xml
<?xml version="1.0"?>
<wsp:Policy
 xmlns:wssp="http://www.bea.com/wls90/security/policy"
 xmlns:wsu="http://docs.oasis-open.org/wss/2004/01/oasis-2004-01-wss-wssecurity-utility-1.0.xsd"
 xmlns:wls="http://www.bea.com/wls90/security/policy/wsee#part"
>
  <wssp:Identity>
    <wssp:SupportedTokens>
      <wssp:SecurityToken TokenType="http://docs.oasis-open.org/wss/2004/01/oasis-2004-01-saml-token-profile-1.0#SAMLAssertionID">
        <wssp:Claims>
          <wssp:ConfirmationMethod>sender-vouches</wssp:ConfirmationMethod>
        </wssp:Claims>
      </wssp:SecurityToken>
    </wssp:SupportedTokens>
  </wssp:Identity>
  <wssp:Confidentiality>
    <wssp:KeyWrappingAlgorithm URI="http://www.w3.org/2001/04/xmlenc#rsa-1_5"/>
  </wssp:Confidentiality>
</wsp:Policy>
```
Element Description

The web service reliable messaging policy assertions in a WS-Policy file are arranged in a specific hierarchy of elements. The hierarchy is shown below. Each element is described in more detail in the sections that follow.

```
Policy [1]
  Identity [1]
    SupportedTokens [0 or 1]
    SecurityToken [1 or more]
      Claims [0 or 1]
        UsePassword [0 or 1]
        ConfirmationMethod [0 or 1]
        TokenLifeTime [0 or 1]
        Length [0 or 1]
        Label [0 or 1]
  Integrity [1]
    SignatureAlgorithm [1]
    CanonicalizationAlgorithm [1]
    SupportedTokens [0 or 1]
    SecurityToken [1 or more]
    Target [1 or more]
      DigestAlgorithm [1]
      Transform [0 or more]
      MessageParts [1]
  Confidentiality [1]
    KeyWrappingAlgorithm [1]
    Target [1 or more]
      EncryptionAlgorithm [1]
      Transform [0 or more]
      MessageParts [1]
    KeyInfo [1]
      SecurityToken [0 or more]
      SecurityTokenReference [0 or more]
    MessageAge [1]
```

CanonicalizationAlgorithm

Specifies the algorithm used to canonicalize the SOAP message elements that are digitally signed.
Note:

The WebLogic web services security runtime does not support specifying an InclusiveNamespaces PrefixList that contains a list of namespace prefixes or a token indicating the presence of the default namespace to the canonicalization algorithm.

Table 6-1  Attributes of <CanonicalizationAlgorithm>

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
<th>Required?</th>
</tr>
</thead>
<tbody>
<tr>
<td>URI</td>
<td>The algorithm used to canonicalize the SOAP message being signed. You can specify only the following canonicalization algorithm: <a href="http://www.w3.org/2001/10/xml-exc-c14n#">http://www.w3.org/2001/10/xml-exc-c14n#</a></td>
<td>Yes</td>
</tr>
</tbody>
</table>

Claims

Specifies additional metadata information that is associated with a particular type of security token. Depending on the type of security token, you can or must specify the following child elements:

- For username tokens, you can define a <UsePassword> child element to specify whether you want the SOAP messages to use password digests. For more information, see UsePassword.
- For SAML tokens, you must define a <ConfirmationMethod> child element to specify the type of SAML confirmation (sender-vouches or holder-of-key). For more information, see ConfirmationMethod.

By default, a security token for a secure conversation has a lifetime of 12 hours. To change this default value, define a <TokenLifeTime> child element to specify a new lifetime, in milliseconds, of the security token. For more information, see TokenLifeTime.

This element does not have any attributes.

Confidentiality

Specifies that part or all of the SOAP message must be encrypted, as well as the algorithms and keys that are used to encrypt the SOAP message.

For example, a web service may require that the entire body of the SOAP message must be encrypted using triple-DES.

Table 6-2  Attributes of <Confidentiality>

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
<th>Required?</th>
</tr>
</thead>
<tbody>
<tr>
<td>SupportTrust10</td>
<td>The valid values for this attribute are true and false. The default value is false.</td>
<td>No</td>
</tr>
</tbody>
</table>
ConfirmationMethod

Specifies the type of confirmation method that is used when using SAML tokens for identity. You must specify one of the following two values for this element: sender-vouches or holder-of-key. For example:

```xml
<wssp:Claims>
    <wssp:ConfirmationMethod>sender-vouches</wssp:ConfirmationMethod>
</wssp:Claims>
```

This element does not have any attributes.

The `<ConfirmationMethod>` element is required only if you are using SAML tokens.

The exact location of the `<ConfirmationMethod>` assertion in the security policy file depends on the type configuration method you are configuring. In particular:

**sender-vouches:**

Specify the `<ConfirmationMethod>` assertion within an `<Identity>` assertion, as shown in the following example:

```xml
<?xml version="1.0"?>
<wsp:Policy
    xmlns:wssp="http://www.bea.com/wls90/security/policy"
    xmlns:wsu="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-utility-1.0.xsd"
    xmlns:wls="http://www.bea.com/wls90/security/policy/wsee#part">
    <wssp:Identity>
        <wssp:SupportedTokens>
            <wssp:SecurityToken
                TokenType="http://docs.oasis-open.org/wss/2004/01/oasis-2004-01-saml-token-profile-1.0#SAMLAssertionID">
                <wssp:Claims>
                    <wssp:ConfirmationMethod>sender-vouches</wssp:ConfirmationMethod>
                </wssp:Claims>
            </wssp:SecurityToken>
        </wssp:SupportedTokens>
    </wssp:Identity>
</wsp:Policy>
```

**holder-of-key:**

Specify the `<ConfirmationMethod>` assertion within an `<Integrity>` assertion. The reason you put the SAML token in the `<Integrity>` assertion for this confirmation method is that the web service runtime must prove the integrity of the message, which is not required by sender-vouches.

For example:

```xml
<?xml version="1.0"?>
<wsp:Policy
    xmlns:wssp="http://www.bea.com/wls90/security/policy"
    xmlns:wsu="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-utility-1.0.xsd"
    xmlns:wls="http://www.bea.com/wls90/security/policy/wsee#part">
    <wssp:Integrity>
```

**Chapter 6**

**Element Description**

6-5
For more information about the two SAML confirmation methods (sender-vouches or holder-of-key), see SAML Token Profile Support in WebLogic Web Services in Understanding Security for Oracle WebLogic Server.

**DigestAlgorithm**

Specifies the digest algorithm that is used when digitally signing the specified parts of a SOAP message. Use the `<MessageParts>` sibling element to specify the parts of the SOAP message you want to digitally sign. For more information, see `MessageParts`.

**Table 6-3  Attributes of `<DigestAlgorithm>`**

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
<th>Required?</th>
</tr>
</thead>
<tbody>
<tr>
<td>URI</td>
<td>The digest algorithm that is used when digitally signing the specified parts of a SOAP message. You can specify only the following digest algorithm: <a href="http://www.w3.org/2000/09/xmldsig#sha1">http://www.w3.org/2000/09/xmldsig#sha1</a></td>
<td>Yes</td>
</tr>
</tbody>
</table>

**EncryptionAlgorithm**

Specifies the encryption algorithm that is used when encrypting the specified parts of a SOAP message. Use the `<MessageParts>` sibling element to specify the parts of the SOAP message you want to digitally sign. For more information, see `MessageParts`. 
Table 6-4  Attributes of <EncryptionAlgorithm>

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
<th>Required?</th>
</tr>
</thead>
<tbody>
<tr>
<td>URI</td>
<td>The encryption algorithm used to encrypt specified parts of the SOAP message. Valid values are: <a href="http://www.w3.org/2001/04/xmlenc#tripledes-cbc">http://www.w3.org/2001/04/xmlenc#tripledes-cbc</a> <a href="http://www.w3.org/2001/04/xmlenc#kw-tripledes">http://www.w3.org/2001/04/xmlenc#kw-tripledes</a> <a href="http://www.w3.org/2001/04/xmlenc#aes128-cbc">http://www.w3.org/2001/04/xmlenc#aes128-cbc</a></td>
<td>Yes</td>
</tr>
</tbody>
</table>

When interoperating between web services built with WebLogic Workshop 8.1, you must specify http://www.w3.org/2001/04/xmlenc#aes128-cbc as the encryption algorithm.

Identity

Specifies the type of security tokens (username, X.509, or SAML) that are supported for authentication.

This element has no attributes.

Integrity

Specifies that part or all of the SOAP message must be digitally signed, as well as the algorithms and keys that are used to sign the SOAP message.

For example, a web service may require that the entire body of the SOAP message must be digitally signed and only algorithms using SHA1 and an RSA key are accepted.

Table 6-5  Attributes of <Integrity>

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
<th>Required?</th>
</tr>
</thead>
<tbody>
<tr>
<td>SignToken</td>
<td>Specifies whether the security token, specified using the &lt;SecurityToken&gt; child element of &lt;Integrity&gt;, should also be digitally signed, in addition to the specified parts of the SOAP message. The valid values for this attribute are true and false. The default value is true.</td>
<td>No</td>
</tr>
<tr>
<td>SupportTrust10</td>
<td>The valid values for this attribute are true and false. The default value is false.</td>
<td>No</td>
</tr>
</tbody>
</table>
Table 6-5  (Cont.) Attributes of <Integrity>

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
<th>Required?</th>
</tr>
</thead>
<tbody>
<tr>
<td>X509AuthConditional</td>
<td>Whenever an Identity assertion includes X.509 tokens in the supported token list, your policy must also have an Integrity assertion. The server will not accept X.509 tokens as proof of authentication unless the token is also used in a digital signature. If the Identity assertion accepts other token types, you may use the X509AuthConditional attribute of the Integrity assertion to specify that the digital signature is required only when the actual authentication token is an X.509 token. Remember that abstract Identity assertions are pre-processed at deploy time and converted into concrete assertions by inserting a list of all token types supported by your runtime environment.</td>
<td>No</td>
</tr>
</tbody>
</table>

KeyInfo

Used to specify the security tokens that are used for encryption.

This element has no attributes.

KeyWrappingAlgorithm

Specifies the algorithm used to encrypt the message encryption key.

Table 6-6  Attributes of <KeyWrappingAlgorithm>

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
<th>Required?</th>
</tr>
</thead>
</table>
| URI       | The algorithm used to encrypt the SOAP message encryption key. Valid values are:  

- http://www.w3.org/2001/04/xmlenc#rsa-1_5  
  (to specify the RSA-v1.5 algorithm)  
- http://www.w3.org/2001/04/xmlenc#rsa-oaep-mgflp  
  (to specify the RSA-OAEP algorithm) | Yes        |

Label

Specifies a label for the security context token. Used when configuring WS-SecureConversation security contexts.

This element has no attributes.

Length

Specifies the length of the key when using security context tokens and derived key tokens. This assertion only applies to WS-SecureConversation security contexts.

The default value is 32.
MessageAge

Specifies the acceptable time period before SOAP messages are declared stale and discarded.

When you include this security assertion in your security policy file, the web services runtime adds a `<Timestamp>` header to the request or response SOAP message, depending on the direction (inbound, outbound, or both) to which the security policy file is associated. The `<Timestamp>` header indicates to the recipient of the SOAP message when the message expires.

For example, assume that your security policy file includes the following `<MessageAge>` assertion:

```
<wsp:Policy
  xmlns:wssp="http://www.bea.com/wls90/security/policy"
  xmlns:wsu="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-utility-1.0.xsd"
>
  <wssp:MessageAge Age="300" />
</wsp:Policy>
```

The resulting generated SOAP message will have a `<Timestamp>` header similar to the following excerpt:

```
<wsu:Timestamp
  wsu:Id="Dy2PFsX3ZQacqNKEANpXbNMnMhmzBmGOA2WDc2E0JpiaaTmbYNwT"
  xmlns:wsu="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-utility-1.0.xsd">
  <wsu:Created>2005-11-09T17:46:55Z</wsu:Created>
</wsu:Timestamp>
```

In the example, the recipient of the SOAP message discards the message if received after 2005-11-09T17:51:55Z, or five minutes after the message was created.

The web services runtime, when generating the SOAP message, sets the `<Created>` header to the time when the SOAP message was created and the `<Expires>` header to the creation time plus the value of the `Age` attribute of the `<MessageAge>` assertion.

The following table describes the attributes of the `<MessageAge>` assertion.

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
<th>Required?</th>
</tr>
</thead>
<tbody>
<tr>
<td>Age</td>
<td>Specifies the actual maximum age time-out for a SOAP message, in seconds.</td>
<td>No</td>
</tr>
</tbody>
</table>

The following table lists the properties that describe the timestamp behavior of the WebLogic web services security runtime, along with their default values.
Table 6-8  Timestamp Behavior Properties

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
<th>Default Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Clock Synchronized</td>
<td>Specifies whether the web service assumes synchronized clocks.</td>
<td>true</td>
</tr>
<tr>
<td>Clock Precision</td>
<td>If clocks are synchronized, describes the accuracy of the synchronization.</td>
<td>60000 milliseconds</td>
</tr>
<tr>
<td></td>
<td><strong>Note:</strong> This property is deprecated as of release 9.2 of WebLogic web services. Use the Clock Skew property instead. If both properties are set, then Clock Skew takes precedence.</td>
<td></td>
</tr>
<tr>
<td>Clock Skew</td>
<td>Specifies the allowable difference, in milliseconds, between the sender and receiver of the message.</td>
<td>60000 milliseconds</td>
</tr>
<tr>
<td>Lax Precision</td>
<td>Allows you to relax the enforcement of the clock precision property.</td>
<td>false</td>
</tr>
<tr>
<td></td>
<td><strong>Note:</strong> This property is deprecated as of release 9.2 of WebLogic web services. Use the Clock Skew property instead.</td>
<td></td>
</tr>
<tr>
<td>Max Processing Delay</td>
<td>Specifies the freshness policy for received messages.</td>
<td>-1</td>
</tr>
<tr>
<td>Validity Period</td>
<td>Represents the length of time the sender wants the outbound message to be valid.</td>
<td>60 seconds</td>
</tr>
</tbody>
</table>

You typically never need to change the values of the preceding timestamp properties. However, if you do need to, you must use the WebLogic Server Administration Console to create the default_wss web service Security Configuration, if it does not already exist, and then update its timestamp configuration by clicking on the Timestamp tab. See Create a web service security configuration for task information and Domains: Web Services Security: Timestamp in the Oracle WebLogic Server Administration Console Online Help for additional reference information about these timestamp properties.

MessageParts

Specifies the parts of the SOAP message that should be signed or encrypted, depending on the grand-parent of the element. You can use either an XPath 1.0 expression or a set of pre-defined functions within this assertion to specify the parts of the SOAP message.

The MessageParts assertion is always a child of a Target assertion. The Target assertion can be a child of either an Integrity assertion (to specify how the SOAP message is digitally signed) or a Confidentiality assertion (to specify how the SOAP messages are encrypted.)

See Using MessageParts To Specify Parts of the SOAP Messages that Must Be Encrypted or Signed for detailed information about using this assertion, along with a variety of examples.
Table 6-9  Attributes of <MessageParts>

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
<th>Required?</th>
</tr>
</thead>
</table>
| Dialect   | Identifies the dialect used to identify the parts of the SOAP message that should be signed or encrypted. If this attribute is not specified, then XPath 1.0 is assumed. The value of this attribute must be one of the following:  
  • http://www.w3.org/TR/1999/REC-xpath-19991116: Specifies that an XPath 1.0 expression should be used against the SOAP message to specify the part to be signed or encrypted.  
  • http://schemas.xmlsoap.org/2002/12/wsse#part: Convenience dialect used to specify that the entire SOAP body should be signed or encrypted.  
  • http://www.bea.com/wls90/security/policy/wsee#part: Convenience dialect to specify that the WebLogic-specific headers should be signed or encrypted. You can also use this dialect to use QNames to specify the parts of the security header that should be signed or encrypted. See Using MessageParts To Specify Parts of the SOAP Messages that Must Be Encrypted or Signed for examples of using these dialects. | Yes |
Table 6-10  Attributes of <SecurityToken>

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
<th>Required?</th>
</tr>
</thead>
<tbody>
<tr>
<td>DerivedFromTokenType</td>
<td>Specifies what security token it is derived from. For example, a value of &quot;<a href="http://schemas.xmlsoap.org/ws/2005/02/sc/sct">http://schemas.xmlsoap.org/ws/2005/02/sc/sct</a>&quot; specifies that it is derived from an old version of Secure Conversation Token.</td>
<td>No</td>
</tr>
<tr>
<td>IncludeInMessage</td>
<td>Specifies whether to include the token in the SOAP message. Valid values are true or false. The default value of this attribute is false when used in the &lt;Confidentiality&gt; assertion and true when used in the &lt;Integrity&gt; assertion. The value of this attribute is always true when used in the &lt;Identity&gt; assertion, even if you explicitly set it to false.</td>
<td>No</td>
</tr>
<tr>
<td>TokenType</td>
<td>Specifies the type of security token. Valid values are:</td>
<td>Yes</td>
</tr>
<tr>
<td></td>
<td>• <a href="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-x509-token-profile-1.0#X509v3">http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-x509-token-profile-1.0#X509v3</a> (To specify a binary X.509 token)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>• <a href="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-username-token-profile-1.0#UsernameToken">http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-username-token-profile-1.0#UsernameToken</a> (To specify a username token)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>• <a href="http://docs.oasis-open.org/wss/2004/01/oasis-2004-01-saml-token-profile-1.0#SAMLAssertionID">http://docs.oasis-open.org/wss/2004/01/oasis-2004-01-saml-token-profile-1.0#SAMLAssertionID</a> (To specify a SAML token)</td>
<td></td>
</tr>
</tbody>
</table>

SecurityTokenReference

For internal use only.

You should never include this security assertion in your custom security policy file; it is described in this section for informational purposes only. The WebLogic web services runtime automatically inserts this security assertion in the security policy file that is published in the dynamic WSDL of the deployed web service. The security assertion specifies WebLogic Server's public key; the client application that invokes the web service then uses it to encrypt the parts of the SOAP message specified by the security policy file. The web services runtime then uses the server's private key to decrypt the message.

SignatureAlgorithm

Specifies the cryptographic algorithm used to compute the digital signature.
Table 6-11  Attributes of <SignatureAlgorithm>

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
<th>Required?</th>
</tr>
</thead>
<tbody>
<tr>
<td>URI</td>
<td>Specifies the cryptographic algorithm used to compute the signature.</td>
<td>Yes</td>
</tr>
<tr>
<td></td>
<td><strong>Note:</strong> Be sure that you specify an algorithm that is compatible with the certificates you are using in your enterprise.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Valid values are:</td>
<td></td>
</tr>
<tr>
<td></td>
<td><a href="http://www.w3.org/2000/09/xmldsig#rsa-sha1">http://www.w3.org/2000/09/xmldsig#rsa-sha1</a></td>
<td></td>
</tr>
<tr>
<td></td>
<td><a href="http://www.w3.org/2000/09/xmldsig#dsa-sha1">http://www.w3.org/2000/09/xmldsig#dsa-sha1</a></td>
<td></td>
</tr>
</tbody>
</table>

SupportedTokens

Specifies the list of supported security tokens that can be used for authentication, encryption, or digital signatures, depending on the parent element.

This element has no attributes.

Target

Encapsulates information about which targets of a SOAP message are to be encrypted or signed, depending on the parent element.

The child elements also depend on the parent element; for example, when used in <Integrity>, you can specify the <DigestAlgorithm>, <Transform>, and <MessageParts> child elements. When used in <Confidentiality>, you can specify the <EncryptionAlgorithm>, <Transform>, and <MessageParts> child elements.

You can have one or more targets.

Table 6-12  Attributes of <Target>

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
<th>Required?</th>
</tr>
</thead>
<tbody>
<tr>
<td>encryptContentOnly</td>
<td>Specifies whether to encrypt an entire element, or just its content.</td>
<td>No</td>
</tr>
<tr>
<td></td>
<td>This attribute can be specified only when &lt;Target&gt; is a child element of &lt;Confidentiality&gt;.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Default value of this attribute is true, which means that only the content is encrypted.</td>
<td></td>
</tr>
</tbody>
</table>

TokenLifeTime

Specifies the lifetime, in seconds, of the security context token or derived key token. This element is used only when configuring WS-SecurityConversation security contexts.

The default lifetime of a security token is 12 hours (43,200 seconds).

This element has no attributes.
Transform

Specifies the URI of a transformation algorithm that is applied to the parts of the SOAP message that are signed or encrypted, depending on the parent element.

You can specify zero or more transforms, which are executed in the order they appear in the <Target> parent element.

Table 6-13  Attributes of <Transform>

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
<th>Required?</th>
</tr>
</thead>
</table>
| URI       | Specifies the URI of the transformation algorithm. Valid URIs are:  
  - http://www.w3.org/2000/09/xmldsig#base64 (Base64 decoding transforms)  
  - http://www.w3.org/TR/1999/REC-xpath-19991116 (XPath filtering)  
  For detailed information about these transform algorithms, see XML-Signature Syntax and Processing at http://www.w3.org/TR/xmldsig-core/#sec-TransformAlg. | Yes |

UsePassword

Specifies that whether the plaintext or the digest of the password appear in the SOAP messages. This element is used only with username tokens.

Table 6-14  Attributes of <UsePassword>

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
<th>Required?</th>
</tr>
</thead>
</table>
| Type      | Specifies the type of password. Valid values are:  
  - http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-username-token-profile-1.0#PasswordText: Specifies that cleartext passwords should be used in the SOAP messages.  
  - http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-username-token-profile-1.0#PasswordDigest: Specifies that password digests should be used in the SOAP messages.  
  Note: For backward compatibility reasons, the two preceding URIs can also be specified with an initial "www." For example:  
  - http://www.docs.oasis-open.org/wss/2004/01/oasis-200401-wss-username-token-profile-1.0#PasswordText  
  - http://www.docs.oasis-open.org/wss/2004/01/oasis-200401-wss-username-token-profile-1.0#PasswordDigest | Yes |
Using MessageParts To Specify Parts of the SOAP Messages that Must Be Encrypted or Signed

When you use either the Integrity or Confidentiality assertion in your security policy file, you are required to also use the Target child assertion to specify the targets of the SOAP message to digitally sign or encrypt. The Target assertion in turn requires that you use the MessageParts child assertion to specify the actual parts of the SOAP message that should be digitally signed or encrypted. This section describes various ways to use the MessageParts assertion.

See Example of a Policy File With Security Elements for an example of a complete security policy file that uses the MessageParts assertion within a Confidentiality assertion. The example shows how to specify that the entire body, as well as the Assertion security header, of the SOAP messages should be encrypted.

You use the Dialect attribute of MessageParts to specify the dialect used to identify the SOAP message parts. The WebLogic web services security runtime supports the following three dialects:

- XPath 1.0
- Pre-Defined wsp:Body() Function
- WebLogic-Specific Header Functions

Be sure that you specify a message part that actually exists in the SOAP messages that result from a client invoke of a message-secured web service. If the web services security runtime encounters an inbound SOAP message that does not include a part that the security policy file indicates should be signed or encrypted, then the web services security runtime returns an error and the invoke fails. The only exception is if you use the WebLogic-specific wls:SystemHeader() function to specify that any WebLogic-specific SOAP header in a SOAP message should be signed or encrypted; if the web services security runtime does not find any of these headers in the SOAP message, the runtime simply continues with the invoke and does not return an error.

XPath 1.0

This dialect enables you to use an XPath 1.0 expression to specify the part of the SOAP message that should be signed or encrypted. The value of the Dialect attribute to enable this dialect is http://www.w3.org/TR/1999/REC-xpath-19991116.

You typically want to specify that the parts of a SOAP message that should be encrypted or digitally signed are child elements of either the soap:Body or soap:Header elements. For this reason, Oracle provides the following two functions that take as parameters an XPath expression:

- wsp:GetBody(xpath_expression)—Specifies that the root element from which the XPath expression starts searching is soap:Body.
- wsp:GetHeader(xpath_expression)—Specifies that the root element from which the XPath expression starts searching is soap:Header.

You can also use a plain XPath expression as the content of the MessageParts assertion, without one of the preceding functions. In this case, the root element from which the XPath expression starts searching is soap:Envelope.
The following example specifies that the AddInt part, with namespace prefix n1 and located in the SOAP message body, should be signed or encrypted, depending on whether the parent Target parent is a child of Integrity or Confidentiality assertion:

```xml
<wssp:MessageParts
    Dialect="http://www.w3.org/TR/1999/REC-xpath-19991116"
    xmlns:n1="http://www.bea.com/foo">
    wsp:GetBody(./*n1:AddInt)
</wssp:MessageParts>
```

The preceding example shows that you should define the namespace of a part specified in the XPath expression (n1 in the example) as an attribute to the MessageParts assertion, if you have not already defined the namespace elsewhere in the security policy file.

The following example is similar, except that the part that will be signed or encrypted is wsu:Timestamp, which is a child element of wsee:Security and is located in the SOAP message header:

```xml
<wssp:MessageParts
    Dialect="http://www.w3.org/TR/1999/REC-xpath-19991116">
    wsp:GetHeader(./*wsee:Security/*wsu:Timestamp)
</wssp:MessageParts>
```

In the preceding example, it is assumed that the wsee: and wse: namespaces have been defined elsewhere in the security policy file.

---

**Note:**

It is beyond the scope of this document to describe how to create XPath expressions. For detailed information, see the XML Path Language (XPath), Version 1.0, at [http://www.w3.org/TR/xpath](http://www.w3.org/TR/xpath) specification.

---

### Pre-Defined wsp:Body() Function

The XPath dialect described in **XPath 1.0** is flexible enough for you to pinpoint any part of the SOAP message that should be encrypted or signed. However, sometimes you might just want to specify that the *entire* SOAP message body be signed or encrypted. In this case using an XPath expression is unduly complicated, so Oracle recommends you use the dialect that pre-defines the wsp:Body() function for just this purpose, as shown in the following example:

```xml
<wssp:MessageParts
    Dialect="http://schemas.xmlsoap.org/2002/12/wse#part">
    wsp:Body()
</wssp:MessageParts>
```

### WebLogic-Specific Header Functions

Oracle provides its own dialect that pre-defines a set of functions to easily specify that some or all of the WebLogic security or system headers should be signed or encrypted. Although you can achieve the same goal using the XPath dialect, it is much
simpler to use this WebLogic dialect. You enable this dialect by setting the Dialect attribute to http://www.bea.com/wls90/security/policy/wsee#part.

The wls:SystemHeaders() function specifies that all of the WebLogic-specific headers should be signed or encrypted. These headers are used internally by the WebLogic web services runtime for various features, such as reliable messaging and addressing. The headers are:

- wsrn:SequenceAcknowledgement
- wsrn:AckRequested
- wsrn:Sequence
- wsa:Action
- wsa:FaultTo
- wsa:From
- wsa:MessageID
- wsa:RelatesTo
- wsa:ReplyTo
- wsa:To
- wsax:SetCookie

The following example shows how to use the wls:SystemHeader() function:

```xml
<wssp:MessageParts Dialect="http://www.bea.com/wls90/security/policy/wsee#part">
  wls:SystemHeaders()
</wssp:MessageParts>
```

Use the wls:SecurityHeader(header) function to specify a particular part in the security header that should be signed or encrypted, as shown in the following example:

```xml
<wssp:MessageParts Dialect="http://www.bea.com/wls90/security/policy/wsee#part">
  wls:SecurityHeader(wsa:From)
</wssp:MessageParts>
```

In the example, only the wsa:From security header is signed or encrypted. You can specify any of the preceding list of headers to the wls:SecurityHeader() function.
WebLogic Web Service Deployment Descriptor Schema Reference

The WebLogic equivalent to the standard Java EE webservices.xml deployment descriptor file is called weblogic-webservices.xml. This file contains WebLogic-specific information about a WebLogic web service, such as the URL used to invoke the deployed web service, configuration settings such as timeout values, and so on.

- Overview of weblogic-webservices.xml
- Example of a weblogic-webservices.xml Deployment Descriptor File
- Element Descriptions

Overview of weblogic-webservices.xml

The standard Java EE deployment descriptor for web services is called webservices.xml. This file specifies the set of web services that are to be deployed to WebLogic Server and the dependencies they have on container resources and other services. See the web services XML Schema at http://java.sun.com/xml/ns/j2ee/j2ee_web_services_1_1.xsd for a full description of this file.

For the XML Schema file that describes the weblogic-webservices.xml deployment descriptor, see http://xmlns.oracle.com/weblogic/weblogic-webservices/1.1/weblogic-webservices.xsd.

Both deployment descriptor files are located in the same location on the Java EE archive that contains the web service. In particular:

- For Java class-implemented web services, the web service is packaged as a Web application WAR file and the deployment descriptors are located in the WEB-INF directory.
- For stateless session EJB-implemented web services, the web service is packaged as an EJB JAR file and the deployment descriptors are located in the META-INF directory.

The structure of the weblogic-webservices.xml file is similar to the structure of the Java EE webservices.xml file in how it lists and identifies the web services that are contained within the archive. For example, for each web service in the archive, both files have a <webservice-description> child element of the appropriate root element (<webservices> for the Java EE webservices.xml file and <weblogic-webservices> for the weblogic-webservices.xml file).

This section is published for informational purposes only. Typically, configuration updates are made using the WebLogic Server Administration Console or using JWS annotations and you will not need to edit either of the deployment descriptor files directly.
Note:

The data type definitions of two elements in the `weblogic-webservices.xml` file (login-config and transport-guarantee) are imported from the Java EE Schema for the `web.xml` file. See the Servlet Deployment Descriptor Schema at [http://java.sun.com/xml/ns/j2ee/web-app_2_4.xsd](http://java.sun.com/xml/ns/j2ee/web-app_2_4.xsd) for details about these elements and data types.

Example of a `weblogic-webservices.xml` Deployment Descriptor File

Learn about the deployment elements that are contained within the `weblogic-webservices.xml` deployment descriptor.

```xml
<?xml version='1.0' encoding='UTF-8'?>
<weblogic-webservices
 xmlns="http://xmlns.oracle.com/weblogic/weblogic-webservices">
<webservice-description>
<webservice-description-name>MyService</webservice-description-name>
<port-component>
<port-component-name>MyServiceServicePort</port-component-name>
<service-endpoint-address>
<webservice-contextpath>/MyService</webservice-contextpath>
<webservice-serviceuri>/MyService</webservice-serviceuri>
</service-endpoint-address>
<wsat-config>
<version>WSAT10</version>
<flowType>SUPPORTS</flowType>
</wsat-config>
<reliability-config>
<inactivity-timeout>P0DT600S</inactivity-timeout>
<base-retransmission-interval>P0DT3S</base-retransmission-interval>
<retransmission-exponential-backoff>true
<retransmission-exponential-backoff>
<acknowledgement-interval>P0DT3S</acknowledgement-interval>
<sequence-expiration>P1D</sequence-expiration>
<buffer-retry-count>3</buffer-retry-count>
<buffer-retry-delay>P0DT5S</buffer-retry-delay>
</reliability-config>
</port-component>
</webservice-description>
</weblogic-webservices>
```

Element Descriptions

The configuration elements specified in the `weblogic-webservices.xml` deployment descriptor are arranged in a specific hierarchy. The hierarchy is shown below. The number of occurrences allowed of each element is identified within braces after the element name. Each element is described in detail in the sections that follow.

```xml
<weblogic-webservices> {1}
  <webservice-description> {1 or more}
    <webservice-description-name> {1 or more}
    <webservice-type> {0 or 1}
```
Chapter 7
Element Descriptions

<wSDL-publish-file> {0 or 1}
<port-component> {0 or more}
  <port-component-name> {1}
  <service-endpoint-address> {0 or 1}
    <webservice-contextpath> {1}
    <webservice-serviceuri> {1}
  <auth-constraint> {0 or 1}
  <login-config> {0 or 1}
  <transport-guarantee> {0 or 1}
  <deployment-listener-list> {0 or 1}
    <deployment-listener> {1 or more}
  <wsdl> {0 or 1}
    <exposed> {1}
    <transaction-timeout> {0 or 1}
    <callback-protocol> {1}
    <stream-attachments> {0 or 1}
    <validate-request> {0 or 1}
    <http-flush-response> {0 or 1}
    <http-response-buffersize> {0 or 1}
  <reliability-config> {0 or 1}
    <customized> {0 or 1}
      <inactivity-timeout> {0 or 1}
      <base-retransmission-interval> {0 or 1}
      <retransmission-exponential-backoff> {0 or 1}
      <non-buffered-source> {0 or 1}
      <acknowledgement-interval> {0 or 1}
      <sequence-expiration> {0 or 1}
      <buffer-retry-count> {0 or 1}
      <buffer-retry-delay> {0 or 1}
      <non-buffered-destination> {0 or 1}
      <messaging-queue-jndi-name> {0 or 1}
      <messaging-queue-mdb-run-as-principal-name> {0 or 1}
  <persistence-config> {0 or 1}
    <customized> {0 or 1}
      <default-logical-store-name> {0 or 1}
      <buffering-config> {0 or 1}
        <customized> {0 or 1}
          <request-queue> {0 or 1}
            <name> {0 or 1}
              <enabled> {0 or 1}
              <connection-factory-jndi-name> {0 or 1}
              <transaction-enabled> {0 or 1}
          <response-queue> {0 or 1}
            <name> {0 or 1}
              <enabled> {0 or 1}
              <connection-factory-jndi-name> {0 or 1}
              <transaction-enabled> {0 or 1}
            <retry-count> {0 or 1}
            <retry-delay> {0 or 1}
        <wsat-config> {0 or 1}
          <version> {0 or 1}
          <flowType> {0 or 1}
        <operation> {0 or more}
          <name> {0 or 1}
          <wsat-config> {0 or 1}
            <version> {0 or 1}
            <flowType> {0 or 1}
        <soapjms-service-endpoint-address> {0 or 1}
          <lookup-variant> {0 or 1}
          <destination-name> {0 or 1}
          <destination-type> {0 or 1}
acknowledgement-interval

The `<acknowledgement-interval>` child element of the `<reliability-config>` element specifies the maximum interval during which the destination endpoint must transmit a stand-alone acknowledgement.

This value must be a positive value and conform to the XML schema duration lexical format, as follows:

```
PnYnMnDnTnHnMnSn
```

Table 7-1 describes the duration format fields. This value defaults to `P0DT3S` (3 seconds).

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>nY</td>
<td>Number of years (n).</td>
</tr>
<tr>
<td>nM</td>
<td>Number of months (n).</td>
</tr>
<tr>
<td>nD</td>
<td>Number of days (n).</td>
</tr>
<tr>
<td>T</td>
<td>Date and time separator.</td>
</tr>
<tr>
<td>nH</td>
<td>Number of hours (n).</td>
</tr>
<tr>
<td>nM</td>
<td>Number of minutes (n).</td>
</tr>
<tr>
<td>nS</td>
<td>Number of seconds (n).</td>
</tr>
</tbody>
</table>

See Configuring the Acknowledgement Interval in Developing JAX-WS Web Services for Oracle WebLogic Server.

activation-config

The `<activation-config>` child element of the `<soapjms-service-endpoint-address>` element specifies activation configuration properties passed to the JMS
provider. Each property is specified using name-value pairs, separated by semicolons (;). For example: name1=value1;...;nameN=valueN

See Using SOAP Over JMS Transport in Developing JAX-WS Web Services for Oracle WebLogic Server. For a list of valid activation properties, see Configuring JMS Transport Properties in Developing JAX-WS Web Services for Oracle WebLogic Server.

**auth-constraint**

The `<auth-constraint>` element defines the user roles that are permitted access to this resource collection.

The XML Schema data type of the `<j2ee:auth-constraint>` element is `<j2ee:auth-constraintType>`, and is defined in the Java EE Schema that describes the standard web.xml deployment descriptor. For the full reference information, see [http://java.sun.com/xml/ns/j2ee/web-app_2_4.xsd](http://java.sun.com/xml/ns/j2ee/web-app_2_4.xsd).

**base-retransmission-interval**

The `<base-retransmission-interval>` child element of the `<reliability-config>` element specifies the interval of time that must pass before a message is retransmitted to the RM destination. This element can be used in conjunction with the `<retransmission-exponential-backoff>` element to specify the algorithm that is used to adjust the retransmission interval.

If a destination endpoint does not acknowledge a sequence of messages for the time interval specified by `<base-retransmission-interval>`, the exponential backoff algorithm is used for timing successive retransmissions by the source endpoint, should the message continue to go unacknowledged.

This value must be a positive value and conform to the XML schema duration lexical format, as follows:

```
PnYnMnDTnHnMnS
```

Table 7-1 describes the duration format fields. This value defaults to P0DT3S (3 seconds).

See Configuring the Base Retransmission Interval in Developing JAX-WS Web Services for Oracle WebLogic Server.

**binding-version**

The `<binding-version>` child element of the `<soapjms-service-endpoint-address>` element defines the version of the SOAP JMS binding. This value must be set to 1.0 for this release, which equates to org.jvnet.ws.jms.JMSBindingVersion.SOAP_JMS_1_0. This value maps to the SOAPJMS_bindingVersion JMS message property.

See Using SOAP Over JMS Transport in Developing JAX-WS Web Services for Oracle WebLogic Server.
buffer-retry-count

The `<buffer-retry-count>` child element of the `<reliability-config>` element specifies the number of times that the JMS queue on the destination WebLogic Server instance attempts to deliver the message from a client that invokes the reliable operation to the web service implementation. This value defaults to 3.

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

buffer-retry-delay

The `<buffer-retry-delay>` child element of the `<reliability-config>` element specifies the amount of time that elapses between message delivery retry attempts. The retry attempts are between the client’s request message on the JMS queue and delivery of the message to the web service implementation.

This value must be a positive value and conform to the XML schema duration lexical format, as follows:

```
PnMnDTnHnMnS
```

Table 7-1 describes the duration format fields. This value defaults to `P0DT5S` (5 seconds).

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

buffering-config

The `<buffering-config>` element groups together the buffering configuration elements. The child elements of the `<buffering-config>` element specify runtime configuration values such as retry counts and delays.

See Configuring Message Buffering for Web Services in *Developing JAX-WS Web Services for Oracle WebLogic Server*.

callback-protocol

The `<callback-protocol>` child element of the `<port-component>` element specifies the protocol used for callbacks to notify clients of an event. Valid values include: `http`, `https`, or `jms`.

connection-factory-jndi-name

The `<connection-factory-jndi-name>` child element of the `<request-queue>` and `<response-queue>` elements specifies the JNDI name of the connection factory to use for request and response message buffering, respectively.

See Configuring Message Buffering for Web Services in *Developing JAX-WS Web Services for Oracle WebLogic Server*. 
customized

The `<customized>` child element of the `<reliability-config>`, `<persistence-config>`, and `<buffering-config>` is a Boolean flag that specifies whether the configuration has been customized.

default-logical-store-name

The `<default-logical-store-name>` child element of the `<persistence-config>` element defines the name of the default logical store.

See Managing Web Service Persistence in Developing JAX-WS Web Services for Oracle WebLogic Server.

delivery-mode

The `<delivery-mode>` child element of the `<soapjms-service-endpoint-address>` element specifies the delivery mode indicating whether the request message is persistent. Valid values are `org.jvnet.ws.jms.DeliveryMode.PERSISTENT` and `org.jvnet.ws.jms.DeliveryMode.NON_PERSISTENT`. This value defaults to PERSISTENT.

See Using SOAP Over JMS Transport in Developing JAX-WS Web Services for Oracle WebLogic Server.

deployment-listener-list

For internal use only.

deployment-listener

For internal use only.

destination-name

The `<destination-name>` child element of the `<soapjms-service-endpoint-address>` element defines the name of the destination queue or topic. This value defaults to `com.oracle.webservices.jms.RequestQueue`.

See Using SOAP Over JMS Transport in Developing JAX-WS Web Services for Oracle WebLogic Server.

destination-type

The `<destination-type>` child element of the `<soapjms-service-endpoint-address>` element defines the destination type. Valid values are `org.jvnet.ws.jms.JMSDestinationType.QUEUE` or `org.jvnet.ws.jms.JMSDestinationType.TOPIC`. This value defaults to QUEUE.

See Using SOAP Over JMS Transport in Developing JAX-WS Web Services for Oracle WebLogic Server.
enable-http-wsdl-access

The `<enable-http-wsdl-access>` child element of the `<soapjms-service-endpoint-address>` element is a Boolean value that specifies whether to publish the WSDL through HTTP.

See Using SOAP Over JMS Transport in Developing JAX-WS Web Services for Oracle WebLogic Server.

enabled

The `<enabled>` child element of the `<request-queue>` and `<response-queue>` elements specifies whether request and response message buffering is enabled, respectively.

See Configuring Message Buffering for Web Services in Developing JAX-WS Web Services for Oracle WebLogic Server.

exposed

The `<exposed>` child element of the `<wsdl>` element is a boolean attribute indicating whether the WSDL should be exposed to the public when the web service is deployed.

fastinfoset

The `<fastinfoset>` child element of the `<port-component>` element is a Boolean flag that specifies whether Fast Infoset is supported for the web service port component.

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

flowType

The `<flowtype>` child element of the `<wsat-config>` element specifies whether the web service atomic transaction coordination context is passed with the transaction flow. Valid values include: NEVER, SUPPORTS, and MANDATORY. The value defaults to SUPPORTS.

For complete details on the valid values and their meanings, and valid value combinations when configuring web service atomic transactions for an EJB-style web service that uses the @TransactionAttribute annotation, see the Flow Type Values table in Enabling Web Services Atomic Transactions on Web Services in Developing JAX-WS Web Services for Oracle WebLogic Server.

http-flush-response

The `<http-flush-response>` child element of the `<port-component>` element specifies whether or not you want to flush the reliable response. This value defaults to true.
http-response-buffersize

The `<http-response-buffersize>` child element of the `<port-component>` element specifies the size of the reliable response buffer that is used to cache the request on the server. This value defaults to 0.

inactivity-timeout

The `<inactivity-timeout>` child element of the `<reliability-config>` element specifies an inactivity interval. If, during the specified interval, an endpoint (RM source or RM destination) has not received application or control messages, the endpoint may consider the RM sequence to have been terminated due to inactivity.

This value must be a positive value and conform to the XML schema duration lexical format, as follows:

\[ PnYnMnDTnHnMnS \]

*Table 7-1* describes the duration format fields. This value defaults to `P0DT600S` (600 seconds).

See Configuring Inactivity Timeout in *Developing JAX-WS Web Services for Oracle WebLogic Server*.

jndi-connection-factory-name

The `<jndi-connection-factory-name>` child element of the `<soapjms-service-endpoint-address>` element defines the JNDI name of the connection factory that is used to establish a JMS connection. This value defaults to `com.oracle.webservices.jms.ConnectionFactory`.

See Using SOAP Over JMS Transport in *Developing JAX-WS Web Services for Oracle WebLogic Server*.

jndi-context-parameter

The `<jndi-context-parameter>` child element of the `<soapjms-service-endpoint-address>` element defines additional JNDI environment properties. Each property is specified using name-value pairs, separated by semicolons (;). For example:

\[ name1=value1;...;nameN=valueN. \]

JNDI properties. Each property is specified using name-value pairs, separated by semicolons (;). For example: `name1=value1;...;nameN=valueN`.

This property can be specified more than once. Each occurrence of the `jndiContextParameter` property specifies a JNDI property name-value pair to be added to the `java.util.Hashtable` sent to the `InitialContext` constructor for the JNDI provider.

See Using SOAP Over JMS Transport in *Developing JAX-WS Web Services for Oracle WebLogic Server*. 
**jndi-initial-context-factory**

The `<jndi-initial-context-factory>` child element of the `<soapjms-service-endpoint-address>` element defines the name of the initial context factory class used for JNDI lookup. This value defaults to `weblogic.jndi.WLInitialContextFactory`. This value maps to the `java.naming.factory.initial` property.

See Using SOAP Over JMS Transport in *Developing JAX-WS Web Services for Oracle WebLogic Server*.

**jndi-url**

The `<jndi-url>` child element of the `<soapjms-service-endpoint-address>` element defines the JNDI provider URL. This value maps to the `java.naming.provider.url` property. This value defaults to `t3://localhost:7001`.

See Using SOAP Over JMS Transport in *Developing JAX-WS Web Services for Oracle WebLogic Server*.

**logging-level**

The `<logging-level>` child element of the `<port-component>` element sets the logging level for the port component. Valid values include: `SEVERE`, `WARNING`, `INFO`, `CONFIG`, `FINE`, `FINER`, `FINEST`, `ALL`, and `OFF`.

**login-config**

The `<j2ee:login-config>` element specifies the authentication method that should be used, the realm name that should be used for this application, and the attributes that are needed by the form login mechanism.

The XML Schema data type of the `<j2ee:login-config>` element is `<j2ee:login-configType>`, and is defined in the Java EE Schema that describes the standard `web.xml` deployment descriptor. For the full reference information, see [http://java.sun.com/xml/ns/j2ee/web-app_2_4.xsd](http://java.sun.com/xml/ns/j2ee/web-app_2_4.xsd).

**lookup-variant**

The `<lookup-variant>` child element of the `<soapjms-service-endpoint-address>` element defines the method used for looking up the specified destination name. This value must be set to `jndi` to support SOAP over JMS transport.

See Using SOAP Over JMS Transport in *Developing JAX-WS Web Services for Oracle WebLogic Server*.

**mbean-name**

The `<mbean-name>` child element of the `<webservice-security>` element specifies the name of the web service security configuration (specifically an instantiation of the `WebserviceSecurityMBean`) that is associated with the web services described in the deployment descriptor file. The default configuration is called `default_wss`.
The associated 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 digital signatures, and so on.

You must create the security configuration (even the default one) using the WebLogic Server Administration Console before you can successfully invoke the web service.

**Note:**

The web service security configuration described by this element applies to all web services contained in the weblogic-webservices.xml file. The jwsc Ant task always packages a web service in its own JAR or WAR file, so this limitation is not an issue if you always use the jwsc Ant task to generate a web service. However, if you update the weblogic-webservices.xml deployment descriptor manually and add additional web service descriptions, you cannot associate different security configurations to different services.

---

**mdb-per-destination**

The `<mdb-per-destination>` child element of the `<soapjms-service-endpoint-address>` element is a Boolean value that specifies whether to create one listening message-driven bean (MDB) for each requested destination. This value defaults to true.

If set to false, one listening MDB is created for each web service port, and that MDB cannot be shared by other ports.

See Using SOAP Over JMS Transport in Developing JAX-WS Web Services for Oracle WebLogic Server.

**message-type**

The `<message-type>` child element of the `<soapjms-service-endpoint-address>` element specifies message type to use with the request message. A value of BYTES indicates the `javax.jms.BytesMessage` object is used. A value of TEXT indicates `javax.jms.TextMessage` object is used. This value defaults to BYTES.

The web service uses the same message type when sending the response. If the request is received as a BYTES, the reply will be sent as a BYTES.

See Using SOAP Over JMS Transport in Developing JAX-WS Web Services for Oracle WebLogic Server.

**messaging-queue-jndi-name**

The `<messaging-queue-jndi-name>` child element of the `<reliability-config>` element specifies the JNDI name of the destination queue or topic.

See Using Web Services Reliable Messaging in Developing JAX-WS Web Services for Oracle WebLogic Server.
messaging-queue-mdb-run-as-principal-name

The `<messaging-queue-mdb-run-as-principal-name>` child element of the `<reliability-config>` element specifies the principal used to run the listening MDB.

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

name

The `<name>` child element of the `<operation>` element defines the name of the web service operation.

non-buffered-destination

The `<non-buffered-destination>` child element of the `<reliability-config>` element is a Boolean value that specifies whether to disable message buffering on a particular destination server to control whether buffering is used when receiving messages.

See Configuring a Non-buffered Destination for a Web Service in *Developing JAX-WS Web Services for Oracle WebLogic Server*.

non-buffered-source

The `<non-buffered-source>` child element of the `<reliability-config>` element is a Boolean value that specifies whether to disable message buffering on a particular source server to control whether buffering is used when delivering messages. This value should always be set to `false`; message buffering should always be enabled on the source server.

See Configuring a Non-buffered Destination for a Web Service in *Developing JAX-WS Web Services for Oracle WebLogic Server*.

operation

The `<operation>` element defines characteristics of a web service operation. The child elements of the `<operation>` element defines the name and configuration options of the web service operation.

persistence-config

The `<persistence-config>` element groups together the persistence configuration elements. The child elements of the `<persistence-config>` element specify the default logical store.

See Managing Web Service Persistence in *Developing JAX-WS Web Services for Oracle WebLogic Server*. 
**port-component**

The `<port-component>` element is a container of other elements used to describe a web service port. The child elements of the `<port-component>` element specify WebLogic-specific characteristics of the web service port, such as the context path and service URI used to invoke the web service after it has been deployed to WebLogic Server.

**port-component-name**

The `<port-component-name>` child element of the `<port-component>` element specifies the internal name of the WSDL port. The value of this element must be unique for all `<port-component-name>` elements within a single `weblogic-webservices.xml` file.

**priority**

The `<priority>` child element of the `<soapjms-service-endpoint-address>` element defines the JMS priority associated with the request and response message. Specify this value as a positive integer from 0, the lowest priority, to 9, the highest priority. This value defaults to 0.

See Using SOAP Over JMS Transport in *Developing JAX-WS Web Services for Oracle WebLogic Server*.

**reliability-config**

The `<reliability-config>` element groups together the reliable messaging configuration elements. The child elements of the `<reliability-config>` element specify runtime configuration values such as retransmission and timeout intervals for reliable messaging.

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

**reply-to-name**

The `<reply-to-name>` child element of the `<soapjms-service-endpoint-address>` element defines the JNDI name of the JMS destination to which the response message is sent.

For a two-way operation, a temporary response queue is generated by default. Using the default temporary response queue minimizes the configuration that is required. However, in the event of a server failure, the response message may be lost. This property enables the client to use a previously defined, “permanent” queue or topic rather than use the default temporary queue or topic, for receiving replies.

The value maps to the `JMSReplyTo` JMS header in the request message.

See Using SOAP Over JMS Transport in *Developing JAX-WS Web Services for Oracle WebLogic Server*. 
request-queue

The `<request-queue>` child element of the `<buffering-config>` element defines the JNDI name of the connection factory to use for request message buffering. This value defaults to the default JMS connection factory defined by the server.

See Configuring the Request Queue in *Developing JAX-WS Web Services for Oracle WebLogic Server*.

response-queue

The `<response-queue>` child element of the `<buffering-config>` element defines the JNDI name of the connection factory to use for response message buffering. This value defaults to the default JMS connection factory defined by the server.

See Configuring the Response Queue in *Developing JAX-WS Web Services for Oracle WebLogic Server*.

retransmission-exponential-backoff

The `<retransmission-exponential-backoff>` child element of the `<reliability-config>` element is a boolean attribute that specifies whether the message retransmission interval will be adjusted using the exponential backoff algorithm. This element is used in conjunction with the `<base-retransmission-interval>` element.

If a destination endpoint does not acknowledge a sequence of messages for the time interval specified by `<base-retransmission-interval>`, the exponential backoff algorithm is used for timing successive retransmissions by the source endpoint, should the message continue to go unacknowledged.

This value defaults to `false`—the same retransmission interval is used in successive retries, rather than the interval increasing exponentially.

See Configuring the Retransmission Exponential Backoff in *Developing JAX-WS Web Services for Oracle WebLogic Server*.

retry-count

The `<retry-count>` child element of the `<buffering-config>` element defines the number of times that the JMS queue on the invoked WebLogic Server instance attempts to deliver the message to the web service implementation until the operation is successfully invoked. This value defaults to `3`.

See Configuring Message Retry Count and Delay in *Developing JAX-WS Web Services for Oracle WebLogic Server*.

retry-delay

The `<retry-delay>` child element of the `<buffering-config>` element defines the number of times that the JMS queue on the invoked WebLogic Server instance attempts to deliver the message to the web service implementation until the operation is successfully invoked. This value defaults to `3`. 
Amount of time between retries of a buffered request and response. Note, this value is only applicable when RetryCount is greater than 0.

This value must be a positive value and conform to the XML schema duration lexical format, as follows:

\[ P_{nYnMnDTnHnMS} \]

Table 7-1 describes the duration format fields. This value defaults to \( P0DT30S \) (30 seconds).

See Configuring Message Retry Count and Delay in Developing JAX-WS Web Services for Oracle WebLogic Server.

run-as-principal

The \(<\text{run-as-principal}>\) child element of the \(<\text{soapjms-service-endpoint-address}>\) element defines the principal used to run the listening MDB.

See Using SOAP Over JMS Transport in Developing JAX-WS Web Services for Oracle WebLogic Server.

run-as-role

The \(<\text{run-as-role}>\) child element of the \(<\text{soapjms-service-endpoint-address}>\) element defines the role used to run the listening MDB.

See Using SOAP Over JMS Transport in Developing JAX-WS Web Services for Oracle WebLogic Server.

sequence-expiration

The \(<\text{sequence-expiration}>\) child element of the \(<\text{reliability-config}>\) element specifies the expiration time for a sequence regardless of activity.

This value must be a positive value and conform to the XML schema duration lexical format, as follows:

\[ P_{nYnMnDTnHnMS} \]

Table 7-1 describes the duration format fields. This value defaults to \( P1D \) (1 day).

See Configuring the Sequence Expiration in Developing JAX-WS Web Services for Oracle WebLogic Server.

service-endpoint-address

The \(<\text{service-endpoint-address}>\) element groups the WebLogic-specific context path and service URI values that together make up the web service endpoint address, or the URL that invokes the web service after it has been deployed to WebLogic Server.

These values are specified with the \(<\text{webservice-contextpath}>\) and \(<\text{webservice-serviceuri}>\) child elements.
soapjms-service-endpoint-address

The `<soapjms-service-endpoint-address>` element groups the configuration properties for SOAP over JMS transport.

See Using SOAP Over JMS Transport in Developing JAX-WS Web Services for Oracle WebLogic Server.

stream-attachments

The `<stream-attachments>` child element of the `<port-component>` element is a boolean value that specifies whether the WebLogic web services runtime uses streaming APIs when reading the parameters of all methods of the web service. This increases the performance of web service operation invocation, in particular when the parameters are large, such as images.

You cannot use this annotation if you are also using the following features in the same web service:

- Conversations
- Reliable Messaging
- JMS Transport
- A proxy server between the client application and the web service it invokes

target-service

The `<target-service>` child element of the `<soapjms-service-endpoint-address>` element defines the port component name of the web service. This value is used by the service implementation to dispatch the service request. If not specified, the service name from the WSDL or `@javax.jms.WebService` annotation is used.

This value maps to the SOAPJMS_targetService JMS message property.

See Using SOAP Over JMS Transport in Developing JAX-WS Web Services for Oracle WebLogic Server.

time-to-live

The `<time-to-live>` child element of the `<soapjms-service-endpoint-address>` element defines the lifetime, in milliseconds, of the request message. A value of 0 indicates an infinite lifetime. If not specified, the JMS-defined default value (180000) is used.

On the service side, `timeToLive` also specifies the expiration time for each MDB transaction.

See Using SOAP Over JMS Transport in Developing JAX-WS Web Services for Oracle WebLogic Server.
transport-guarantee

The j2ee:transport-guarantee element specifies the type of communication between the client application invoking the web service and WebLogic server.

Valid values include:

- **INTEGRAL**—Application requires that the data sent between the client and server be sent in such a way that it cannot be changed in transit.
- **CONFIDENTIAL**—Application requires that the data be transmitted in a way that prevents other entities from observing the contents of the transmission.
- **NONE**—Application does not require transport guarantees.

The XML Schema data type of the j2ee:transport-guarantee element is j2ee:transport-guaranteeType, and is defined in the Java EE Schema that describes the standard web.xml deployment descriptor. For the full reference information, see http://java.sun.com/xml/ns/j2ee/web-app_2_4.xsd.

transaction-enabled

The <transaction-enabled> child element of the <request-queue> and <response-queue> elements is a Boolean value that specifies whether transactions should be used when storing and retrieving messages from the request and response buffering queues, respectively. This flag defaults to false.

See Configuring Message Buffering for Web Services in Developing JAX-WS Web Services for Oracle WebLogic Server.

transaction-timeout

The <transaction-timeout> child element of the <port-component> element specifies a timeout value for the current transaction, if the web service operation(s) are running as part of a transaction.

This value must be a positive value and conform to the XML schema duration lexical format, as follows:

```
PnYnMnDnTnHnMnSnS
```

Table 7-1 describes the duration format fields. This value defaults to 30 seconds.

validate-request

The <validate-request> child element of the <port-component> element is a boolean value that specifies whether the request should be validated.

The value specified must be a positive value and conform to the XML schema duration lexical format, as follows:

```
PnYnMnDnTnHnMnSnS
```

Table 7-1 describes the duration format fields. This value defaults to P0DT3S (3 seconds).
version

The <version> child element of the <wsat-config> element specifies the version of the web service atomic transaction coordination context that is used for web services and clients. For clients, it specifies the version used for outbound messages only. The value specified must be consistent across the entire transaction.

Valid values include WSAT10, WSAT11, WSAT12, and DEFAULT. The DEFAULT value for web services is all three versions (driven by the inbound request); the DEFAULT value for web service clients is WSAT10.

For more information about web service atomic transactions, see Using Web Service Atomic Transactions in Developing JAX-WS Web Services for Oracle WebLogic Server.

weblogic-webservices

The <weblogic-webservices> element is the root element of the WebLogic-specific web services deployment descriptor (weblogic-webservices.xml).

The element specifies the set of web services contained in the Java EE component archive in which the deployment descriptor is also contained. The archive is either an EJB JAR file (for stateless session EJB-implemented web services) or a WAR file (for Java class-implemented web services).

webservice-contextpath

The <webservice-contextpath> element specifies the context path portion of the URL used to invoke the web service. The URL to invoke a web service deployed to WebLogic Server is:

http://host:port/contextPath/serviceURI

where

- host is the host computer on which WebLogic Server is running.
- port is the port address to which WebLogic Server is listening.
- contextPath is the value of this element
- serviceURI is the value of the webservice-serviceuri element.

When using the jwsc Ant task to generate a web service from a JWS file, the value of the <webservice-contextpath> element is taken from the contextPath attribute of the WebLogic-specific @WLHttpTransport annotation or the <WLHttpTransport> child element of jwsc.

webservice-description

The <webservice-description> element is a container of other elements used to describe a web service. The <webservice-description> element defines a set of port components (specified using one or more <port-component> child elements) that are associated with the WSDL ports defined in the WSDL document.
There may be multiple `<webservice-description>` elements defined within a single `weblogic-webservices.xml` file, each corresponding to a particular stateless session EJB or Java class contained within the archive, depending on the implementation of your web service. In other words, an EJB JAR contains the EJBs that implement a web service, a WAR file contains the Java classes.

**webservice-description-name**

The `<webservice-description-name>` element specifies the internal name of the web service. The value of this element must be unique for all `<webservice-description-name>` elements within a single `weblogic-webservices.xml` file.

**webservice-security**

Element used to group together all the security-related elements of the `weblogic-webservices.xml` deployment descriptor.

**webservice-serviceuri**

The `<webservice-serviceuri>` element specifies the web service URI portion of the URL used to invoke the web service. The URL to invoke a web service deployed to WebLogic Server is:

```
http://host:port/contextPath/serviceURI
```

where

- `host` is the host computer on which WebLogic Server is running.
- `port` is the port address to which WebLogic Server is listening.
- `contextPath` is the value of the `webservice-contextpath` element
- `serviceURI` is the value of this element.

When using the `jwsc` Ant task to generate a web service from a JWS file, the value of the `<webservice-serviceuri>` element is taken from the `serviceURI` attribute of the WebLogic-specific `@WLHttpTransport` annotation (JAX-RPC only) or the `<WLHttpTransport>` child element of `jwsc`.

**webservice-type**

The `<webservice-type>` element specifies whether the web service is based on the JAX-WS or JAX-RPC standard. Valid values include: JAXWS and JAXRPC. This value defaults to JAXRPC.

**wsat-config**

The `<wsat-config>` element enables and configures web service atomic transaction configuration at the class or synchronous method level. The child elements of the `<wsat-config>` element specify the WS-AtomicTransaction version supported and whether or not the web service atomic transaction coordination context is passed with the transaction flow.
For more information about web service atomic transactions, see Using Web Service Atomic Transactions in *Developing JAX-WS Web Services for Oracle WebLogic Server*.

**wsdl**

The `<wsdl>` element groups together all the WSDL-related elements of the `weblogic-webservices.xml` deployment descriptor.

**wsdl-publish-file**

The `<wsdl-publish-file>` element specifies a directory (on the system that hosts the web service) to which WebLogic Server should publish a hard-copy of the WSDL file of a deployed web service; this is in addition to the standard WSDL file accessible via HTTP.

For example, assume that your web service is implemented with an EJB, and its WSDL file is located in the following directory of the EJB JAR file, relative to the root of the JAR:

```
META-INF/wsdl/a/b/Fool.wsdl
```

Further assume that the `weblogic-webservices.xml` file includes the following element for a given web service:

```
<wsdl-publish-file>d:/bar</wsdl-publish-file>
```

This means that when WebLogic Server deploys the web service, the server publishes the WSDL file at the standard HTTP location, but also puts a copy of the WSDL file in the following directory of the computer on which the service is running:

```
d:/bar/a/b/Foo.wsdl
```

**Note:**

Only specify this element if client applications that invoke the web service need to access the WSDL via the local file system or FTP; typically, client applications access the WSDL using HTTP, as described in Browsing to the WSDL of the web service in *Oracle Fusion Middleware Developing JAX-RPC Web Services for Oracle WebLogic Server*.

The value of this element should be an absolute directory pathname. This directory must exist on every machine which hosts a WebLogic Server instance or cluster to which you deploy the web service.