Oracle® Fusion Middleware

WebLogic Web Services Reference for Oracle WebLogic Server

14c (14.1.2.0.0)
F53592-01
December 2024

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

F53592-01

Contents

Preface

This document provides reference information for developing WebLogic web services for Oracle WebLogic Server 14c.

Audience

This documentation is for software developers who are responsible for developing WebLogic web services for Oracle WebLogic Server. It is assumed that the reader is familiar with WebLogic concepts.

Documentation Accessibility

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

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.

Diversity and Inclusion

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

Related Documentation

New and Changed WebLogic Server Features

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

Conventions

The following text conventions are used in this document:

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

1 Introduction

This chapter list 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

This Reference Topic . . .	Describes . . .
Ant Task Reference	WebLogic web services Ant tasks.
JWS Annotation Reference	JWS annotations that you can use in the JWS file that implements your web service.
Web Service Reliable Messaging Policy Assertion Reference	Policy assertions you can add to a WS-Policy file to configure the web service reliable messaging feature of a WebLogic web service.
Web Service MakeConnection Policy Assertion Reference	Policy assertions you can add to a WS-Policy file to configure the web service MakeConnection feature of a WebLogic web service.
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 Deployment Descriptor Schema Reference	Elements in the WebLogic-specific web services deployment descriptor `weblogic-webservices.xml`.

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

2
Ant Task Reference

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.

This chapter includes the following sections:

- Overview of WebLogic Web Services Ant Tasks
- 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.
- 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 JAX-WS.
- wsdlget
  The `wsdlget` Ant task downloads to the local directory a WSDL and its imported XML resources.

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

Ant Task	Description
clientgen	Generates the `Service` stubs and other client-side artifacts used to invoke a web service.
wsdlc	Generates a partial web service implementation based on a WSDL file.
wsdlget	Downloads to the local directory a WSDL and its imported XML targets, such as XSD and WSDL files.

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*

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.

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

- Standalone Jakarta clients that do not use the Jakarta Platform, Enterprise Edition client container.
- Jakarta EE clients, such as EJBs, JSPs, and web services, that use the Jakarta EE client container.

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.

```
<taskdef name="clientgen"
    classname="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 JAX-WS.

For JAX-WS, one or more customization files that specify one or more of the following:

- 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*.
- SOAP handler files. See Creating and Using SOAP Message Handlers in *Developing JAX-WS Web Services for Oracle WebLogic Server*.

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`:

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

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

```
<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` and navigate to the description of the FileSet type.

jmstransportclient

> **Note:**
>
> The `<jmstransportclient>` child element applies to JAX-WS only.

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`.

```
<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>
```

xmlcatalog

> **Note:**
>
> The `<xmlcatalog>` child element applies to JAX-WS only.

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

```
<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:

```
<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 `clientgen`. Relevant code lines are shown in **bold**.

```
<target name="clientgen">
<clientgen
     type="JAXWS"
     wsdl="${wsdl}"
     destDir="${clientclasses.dir}"
```

```
     packageName="xmlcatalog.jaxws.clientgen.client"
     catalog="wsdlcatalog.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 describe the WebLogic-specific attributes of the `clientgen` Ant task for JAX-WS web services.

Table 2-2 WebLogic-specific Attributes of the clientgen Ant Task

Attribute	Description	Data Type	Required?
`catalog`	Specifies an external XML catalog file. See Using XML Catalogs in *Developing JAX-WS Web Services for Oracle WebLogic Server*.	String	No
`copyWsdl`	Controls whether the WSDL should be copied in the destination directory defined by `destDir`.	Boolean	No
`destDir`	Directory into which the `clientgen` Ant task generates the client source code, compiled classes, 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.	String	You must specify either the `destFile` or `destDir` attribute, but not both.
`destFile`	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.	String	You must specify either the `destFile` or `destDir` attribute, but not both.

Table 2-2 (Cont.) WebLogic-specific Attributes of the clientgen Ant Task

Attribute	Description	Data Type	Required?
`failonerror`	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.	Boolean	No
`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
`packageName`	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 `http://example.org`, 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.	String	No
`type`	Specifies the type of web service for which you are generating client artifacts. Valid values is `JAXWS`.	String	No
`wsdl`	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.	String	Yes
`wsdlLocation`	Specifies the value of the `wsdlLocation` attribute generated on the `@WebServiceClient`.	String	No

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 `wsdl` 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-WS web service.

```
<taskdef name="clientgen"
    classname="weblogic.wsee.tools.anttasks.ClientGenTask" />
...
<target name="build_client">
<clientgen
    wsdl="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-WS; the following example shows how to use the `type` attribute to specify that the web service is based on JAX-WS:

```
<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.

```
<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.

```
<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:

```
<clientgen
    type="JAXWS"
    wsdl="http://example.com/myapp/mySecuredService.wsdl"
    destDir="/output/clientclasses"
    packageName="myapp.mysecuredservice.client"
    serviceName="SecureStockQuoteService"
    <sysproperty key="javax.net.ssl.trustStore"
                 value="/keystores/DemoTrust.jks"/>
    <sysproperty key="weblogic.wsee.client.ssl.stricthostchecking"
                 value="false"/>
    <sysproperty key="javax.xml.ws.security.auth.username"
                 value="juliet"/>
    <sysproperty key="javax.xml.ws.security.auth.password"
                 value="secret"/>
</clientgen>
```

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 JAX-WS.

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.
- 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.
- You cannot use any WebLogic-specific JWS annotations in a JAX-WS web service.
- For JAX-WS, you can specify JAX-WS (JSR 224 at `https://jcp.org/en/jsr/detail?id=224`), JAXB (JSR 222 at `http://jcp.org/en/jsr/detail?id=222`), or Common (JSR 250 at `http://jcp.org/en/jsr/detail?id=250`) annotations, as required.

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), 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*.

- 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

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

Child Elements

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

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 the JAX-WS web service.

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*.

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` 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:

```
<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:

```
<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**.

```
<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 2-3 WebLogic-specific Attributes of the wsdlc Ant Task

Attribute	Description	Data Type	Required?
`catalog`	Specifies an external XML catalog file. See Using XML Catalogs in *Developing JAX-WS Web Services for Oracle WebLogic Server*.	String	No
`destImplDir`	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.	String	No

Table 2-3 (Cont.) WebLogic-specific Attributes of the wsdlc Ant Task

Attribute	Description	Data Type	Required?
`destJavadocDir`	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.	String	No
`destJwsDir`	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`.	String	Yes
`explode`	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.	Boolean	No
`packageName`	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 `targetNamespace` of the WSDL.	String	No
`srcPortName`	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 `<port>` element that corresponds to the web service port for which you want to generate a JWS interface file. The `<port>` element is a child element of the `<service>` element in the WSDL file. If you do not specify this attribute, `wsdlc` generates a JWS interface file from the service specified by `srcServiceName`.	String	No

Table 2-3 (Cont.) WebLogic-specific Attributes of the wsdlc Ant Task

Attribute	Description	Data Type	Required?
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 <service> 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 <service> element contains more than one <port> 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 <address> child element. Because WebLogic web services support only three transports (JMS, HTTP, and HTTPS), this means that there can be at most three <port> child elements for the <service> element specified by this attribute. The generated JWS implementation file will then include the corresponding @WLXXXTransport annotations. If you do not specify either this or the srcPortName attribute, the WSDL file must include only *one* <service> element. The wsdlc Ant task generates the JWS interface file and data binding JAR file from this single web service.	String	No
srcWsdl	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.	String	Yes
type	Specifies the type of web service for which you are generating a partial implementation: Valid value is JAXWS.	String	No

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 `wsdlc` Ant task:

- <FileSet>
- <SourcePath>
- <Classpath>
- <Extdirs>

Example

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

```
  <taskdef name="wsdlc"
           classname="weblogic.wsee.tools.anttasks.WsdlcTask"/>
  <taskdef name="jwsc"
    classname="weblogic.wsee.tools.anttasks.JwscTask" />
  <target name="generate-from-wsdl">
    <wsdlc
        srcWsdl="wsdl_files/TemperatureService.wsdl"
        destJwsDir="output/compiledWsdl"
        destImplDir="output/impl"
        packageName="examples.webservices.wsdlc"
        type="JAXWS" />
  </target>
  <target name="build-service">
    <jwsc
      srcdir="src"
      destdir="output/wsdlcEar">
      <jws file=
"examples/webservices/wsdlc/TemperatureService_TemperaturePortTypeImpl.java"
           compiledWsdl="output/compiledWsdl/TemperatureService_wsdl.jar"
           type="JAXWS"/>
    </jwsc>
  </target>
```

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`.

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 `compiledWsdl` 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.

wsdlget

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

```
<taskdef name="wsdlget"
       classname="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:

```
<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:

```
<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**.

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

Table 2-4 WebLogic-specific Attributes of the wsdlget Ant Task

Attribute	Description	Data Type	Required?
`catalog`	Specifies an external XML catalog file. See Using XML Catalogs in *Developing JAX-WS Web Services for Oracle WebLogic Server*.	String	No
`destDir`	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.	String	Yes
`wsdl`	Name of the WSDL to copy to the local directory.	String	No

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.

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

3

JWS Annotation Reference

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

This chapter includes the following sections:

- 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.
- Web Services Metadata Annotations (JSR-181)
  Understand the standard JSR-181 annotations that you can use in your JWS file to specify the shape and behavior of your web service.
- JAX-WS Annotations (JSR-224)
- JAXB Annotations (JSR-222)
- Jakarta 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 the Web Service. The Web Service annotation support for JAX-WS are as follows:

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)

Understand the standard JSR-181 annotations that you can use 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 https://jakarta.ee/specifications/web-services-metadata/2.1/apidocs/javax/jws/package-summary or `javax.jws.soap` package at https://jakarta.ee/specifications/web-services-metadata/2.1/apidocs/javax/jws/soap/package-summary and are described in more detail in the Jakarta Web Services Metadata (JSR-181) specification at http://www.jcp.org/en/jsr/detail?id=181.

Table 3-1 Standard JSR-181 JWS Annotations

This annotation . . .	Specifies . . .
`javax.jws.WebService`	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*.
`javax.jws.WebMethod`	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*.
`javax.jws.OneWay`	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*.
`javax.jws.WebParam`	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*.
`javax.jws.WebResult`	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*.
`javax.jws.HandlerChain`	An external handler chain. For more information, see Creating and Using SOAP Message Handlers in *Developing JAX-WS Web Services for Oracle WebLogic Server*.
`javax.jws.SOAPBinding`	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*.

JAX-WS Annotations (JSR-224)

Understand the JAX-WS (JSR-224) annotations that you can use in your JWS file to specify the shape and behavior of your web service.The following table summarizes these JAX-WS annotations, which are available with the `javax.xml.ws` package at https://javaee.github.io/metro-jax-ws/ and are described in more detail in JSR 224 (JAX-WS)

Annotations at https://javaee.github.io/metro-jax-ws/doc/user-guide/ch03.html#jsr-224-jax-ws-annotations-outline.

> **Note:**
>
> The JAX-WS JWS annotations are relevant to JAX-WS web services only.

Table 3-2 JAX-WS (JSR-224) Annotations

This annotation . . .	Specifies . . .
`javax.xml.ws.Action`	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.
`javax.xml.ws.BindingType`	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*.
`javax.xml.ws.FaultAction`	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.
`javax.xml.ws.RequestWrapper`	The request wrapper bean to be used at runtime for the methods in the endpoint interface.
`javax.xml.ws.ResponseWrapper`	The response wrapper bean to be used at runtime for the methods in the endpoint interface.
`javax.xml.ws.ServiceMode`	Whether a provider implementation works with the entire protocol message or with the payload only.
`javax.xml.ws.WebEndpoint`	The `get`*`PortName`*`()` methods of a generated service interface.
`javax.xml.ws.WebFault`	Service-specific exception classes to customize to the local and namespace name of the fault element and the name of the fault bean.
`javax.xml.ws.WebServiceClient`	A generated service interface.
`javax.xml.ws.WebServiceProvider`	A provider implementation class.
`javax.xml.ws.WebServiceRef`	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*.

JAXB Annotations (JSR-222)

Understand the JAXB (JSR-222) annotations that you can use in your JWS file to specify the shape and behavior of your web service.

The following table summarizes these JAXB annotations, which are available with the `javax.xml.bind.annotation` package at https://jakarta.ee/specifications/xml-binding/3.0/apidocs/jakarta.xml.bind/jakarta/xml/bind/annotation/package-summary. They 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 in the JAXB (JSR-222) specification at http://jcp.org/en/jsr/detail?id=222).

> **Note:**
>
> The JAXB JWS annotations are relevant to JAX-WS Web Services only.

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

This annotation . . .	Specifies . . .
`javax.xml.bind.annotation.XmlAccessorType`	Whether fields or properties are serialized by default. See Specifying Default Serialization of Fields and Properties (@XmlAccessorType) in *Developing JAX-WS Web Services for Oracle WebLogic Server*.
`javax.xml.bind.annotation.XmlElement`	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*.
`javax.xml.bind.annotation.XmlRootElement`	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*.
`javax.xml.bind.annotation.XmlSeeAlso`	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*.
`javax.xml.bind.annotation.XmlType`	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*.

Jakarta Annotations (JSR-250)

Understand the Jakarta Annotations for the Java Platform (JSR-250) that you can use 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 https://jakarta.ee/specifications/annotations/1.3/apidocs/ and are described in more detail in the Jakarta Annotations for the Java Platform (JSR-250) specification at http://jcp.org/en/jsr/detail?id=250.

Table 3-4 Jakarta Annotations (JSR-250)

This annotation . . .	Specifies . . .
`javax.annotation.Resource`	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.
`javax.annotation.PostConstruct`	A method that needs to be executed after dependency injection is done to perform initialization.
`javax.annotation.PreDestroy`	A callback notification om a method to signal that the instance is in the process of being removed by the container.

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 supported for JAX-WS. Each annotation is described in more detail in the sections that follow.

Table 3-5 WebLogic-specific Annotations

This annotation . . .	Specifies . .
com.oracle.webservices.api.jms.JMSTransportClient	That the web service client supports SOAP over JMS transport connection protocol.
com.oracle.webservices.api.jms.JMSTransportService	That the web service supports SOAP over JMS transport connection protocol.
weblogic.jws.Policies	An array of `@weblogic.jws.Policy` annotations.
weblogic.jws.Policy	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.
weblogic.jws.security.WssConfiguration	The name of the Web Service security configuration you want the Web Service to use.
weblogic.wsee.jws.jaxws.owsm.Property	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.
weblogic.wsee.jws.jaxws.owsm.SecurityPolicies	An array of `@weblogic.wsee.jws.jaxws.owsm.SecurityPolicies` annotations.
weblogic.wsee.jws.jaxws.owsm.SecurityPolicy	That an Oracle Web Services Manager (OWSM) security policy be attached to the web service or client.
weblogic.wsee.jws.jaxws.owsm.SecurityPolicies	An array of `@weblogic.jws.SecurityPolicy` annotations.
weblogic.wsee.jws.jaxws.owsm.SecurityPolicy	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.
weblogic.wsee.wstx.wsat.Transactional	Whether the annotated class or method runs inside of a web service atomic transaction.

- com.oracle.webservices.api.jms.JMSTransportClient
- com.oracle.webservices.api.jms.JMSTransportService
- weblogic.jws.Policies
- weblogic.jws.Policy
- weblogic.jws.security.WssConfiguration
- weblogic.wsee.jws.jaxws.owsm.Property
- weblogic.wsee.jws.jaxws.owsm.SecurityPolicies
- weblogic.wsee.jws.jaxws.owsm.SecurityPolicy
- weblogic.wsee.jws.jaxws.owsm.SecurityPolicies
- weblogic.wsee.jws.jaxws.owsm.SecurityPolicy
- weblogic.wsee.wstx.wsat.Transactional

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.

```
...
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.

```
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",
   targetNamespace="http://examples.org/")
@JMSTransportService(destinationName="myQueue",
   activationConfig = {
      @ActivationConfigProperty(
         propertyName  = "destinationType",
         propertyValue = "javax.jms.Topic"),
      @ActivationConfigProperty(
         propertyName  = "subscriptionDurability",
         propertyValue = "Durable"),
      @ActivationConfigProperty(propertyName  = "topicMessagesDistributionMode",
         propertyValue = "One-Copy-Per-Application")})
@BindingType(SOAPBindingImpl.SOAP11_JMS_BINDING)
public class NotifyServiceImpl {..}
```

weblogic.jws.Policies

The following sections describe the annotation in detail.

- Description
- Example

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

```
@Policies({
    @Policy(uri="policy:firstPolicy.xml"),
    @Policy(uri="policy:secondPolicy.xml")
  })
```

weblogic.jws.Policy

The following sections describe the annotation in detail.

- Description
- Attributes

- Example

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.

Attributes

Table 3-6 Attributes of the weblogic.jws.Policy JWS Annotation Tag

Name	Description	Data Type	Required?
`uri`	Specifies the location from which to retrieve the WS-Policy file. Use the `http:` prefix to specify the URL of a WS-Policy file on the Web. Use the `policy:` prefix to specify that the WS-Policy file is packaged in the Web Service archive file or in a shareable Jakarta EE library of WebLogic Server, as shown in the following example: `@Policy(uri="policy:MyPolicyFile.xml")` 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 `META-INF/policies` or `WEB-INF/policies` 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 Jakarta EE Libraries and Optional Packages in *Developing Applications for Oracle WebLogic Server*.	String	Yes

Table 3-6 (Cont.) Attributes of the weblogic.jws.Policy JWS Annotation Tag

Name	Description	Data Type	Required?
`direction`	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 default value is `Policy.Direction.both`.	enum	No
`attachToWsdl`	Specifies whether the WS-Policy file should be attached to the WSDL that describes the Web Service. Valid values are `true` and `false`. Default value is `false`.	boolean	No

Example

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

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 user name/password token), then the Web Service must be associated with a security configuration.

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 3-7 Attributes of the weblogic.jws.security.WssConfiguration JWS Annotation Tag

Name	Description	Data Type	Required?
`value`	Specifies the name of the Web Service security configuration that is associated with this Web Service. The default configuration is called `default_wss`.	String	Yes

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:

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

weblogic.wsee.jws.jaxws.owsm.Property

The following sections describe the annotation in detail.

- Description
- Example

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

```
@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
- Example

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

```
@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
- Attributes

- Examples

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 3-8 Attributes of the weblogic.wsee.jws.jaxws.owsm.SecurityPolicy JWS Annotation Tag

Name	Description	Data Type	Required?
`uri`	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="policy:oracle/wss_saml20_token_over_ssl_service_policy")` For more information about the OWSM repository, see Managing the OWSM Repository in *Securing Web Services and Managing Policies with Oracle Web Services Manager*.	String	Yes
`properties`	**Note**: 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.	String	No
`enabled`	Specifies whether the OWSM policy file is enabled. Valid values are `true` and `false`. Default value is `true`.	boolean	No

Examples

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

```
@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.

```
@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
- Example

Description

Target: Class, Method

Specifies an array of `@weblogic.wsee.jws.jaxws.owsm.SecurityPolicy` annotations.

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

```
@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.

- Description
- Attribute
- Example

Description

Target: Class, Method

Specifies 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.

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

This annotation can be applied at the class level only, indicating that the Oracle WSM WS-Policy file or files are applied to every public operation of the Web Service.

The Oracle WSM WS-Security policies are not advertised in the WSDL of a WebLogic Server JAX-WS Web service. (Typically, the policy file associated with a Web service is attached to its WSDL, which the Web services client runtime reads to determine whether and how to digitally sign and encrypt the SOAP message request from an operation invoke from the client application.)

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.

Attribute

Table 3-9 Attribute of the weblogic.jws.SecurityPolicy JWS Annotation Tag

Name	Description	Data Type	Required?
`uri`	Specifies the location from which to retrieve the Oracle WSM WS-Policy file. Use the `http:` prefix to specify the URL of an Oracle WSM WS-Policy file on the Web. Use the `policy:` prefix to specify that the Oracle WSM WS-Policy file is packaged in the Web Service archive file or in a shareable Jakarta EE library of WebLogic Server, as shown in the following example: `@SecurityPolicy(uri= "policy:oracle/wss10_username_token_with_message_protection_server_policy")`	String	Yes

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
- Attributes
- Example

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 3-10 Attribute of the weblogic.wsee.wstx.wsat.Transactional Annotation

Name	Description	Data Type	Required?
`version`	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])`	String	No
`value`	Whether the web service atomic transaction coordination context is passed with the transaction flow. For valid values, see Table 3-11.	String	No

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 3-11 Flow Types Values

Value	Web Service Client	Web Service	Valid EJB `@TransactionAttribute` Values
`NEVER`	Do not export transaction coordination context.	Do not import transaction coordination context.	`NEVER`, `NOT_SUPPORTED`, `REQUIRED`, `REQUIRES_NEW`, `SUPPORTS`
`SUPPORTS` (Default)	Export transaction coordination context if transaction is available.	Import transaction coordination context if available in the message.	`REQUIRED`, `SUPPORTS`
`MANDATORY`	Export transaction coordination context. An exception is thrown if there is no active transaction.	Import transaction coordination context. An exception is thrown if there is no active transaction.	`MANDATORY`, `REQUIRED`, `SUPPORTS`

Example

```
@Transactional(value = Transactional.TransactionFlowType.SUPPORTS,
     version="Transactional.Versino.WSAT12
```

4

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.

This chapter includes the following sections:

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/`. 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 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 creating a reliable WebLogic web service, see Using Web Services Reliable Messaging in *Developing JAX-WS 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

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

```
<?xml version="1.0"?>
<wsp:Policy xmlns:wsp="http://schemas.xmlsoap.org/ws/2004/09/policy">
    <wsrmp:RMAssertion
           xmlns:wsrmp="http://docs.oasis-open.org/ws-rx/wsrmp/200702">
        <wsrmp:SequenceSTR/>
        <wsrmp:DeliveryAssurance>
          <wsp:Policy>
            <wsrmp:ExactlyOnce/>
          </wsp:Policy>
        </wsrmp:DeliveryAssurance>
    </wsrmp:RMAssertion>
</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
  wsrmp:RMAssertion
    wsrmp:SequenceSTR
    wsrmp:SequenceTransportSecurity
    wsrmp:DeliveryAssurance
      wsp:Policy

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

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

Delivery Assurance	Description
`wsrmp:AtMostOnce`	Messages are delivered at most once, without duplication. It is possible that some messages may not be delivered at all.
`wsrmp:AtLeastOnce`	Every message is delivered at least once. It is possible that some messages are delivered more than once.
`wsrmp:ExactlyOnce`	Every message is delivered exactly once, without duplication.This value is enabled by default.
`wsrmp:InOrder`	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.

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

```
<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>

Attribute	Description	Required?
`optional`	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`.	No

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/.

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 version="1.0"?>
<wsp:Policy
   xmlns:wsrm="http://schemas.xmlsoap.org/ws/2005/02/rm/policy"
   xmlns:wsp="http://schemas.xmlsoap.org/ws/2004/09/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.

> **Note:**
>
> 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>

Attribute	Description	Required?
`Expires`	The amount of time after which the reliable web service expires. The format of this attribute conforms to the XML Schema duration at `http://www.w3.org/TR/2001/REC-xmlschema-2-20010502/#duration` data type. For example, to specify that the reliable web service expires after 3 hours, specify `Expires="P3H"`.	Yes

beapolicy:QOS

Specifies the delivery assurance (or *Quality Of Service*) of the web service:

Table 4-4 Attributes of <beapolicy:QOS>

Attribute	Description	Required?
`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>

Attribute	Description	Required?
`Milliseconds`	Specifies the maximum interval, in milliseconds, in which the destination endpoint must transmit a stand alone acknowledgement.	Yes

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 acknowledgment 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.

Table 4-6 Attributes of <wsrm:BaseRetransmissionInterval>

Attribute	Description	Required?
`Milliseconds`	Number of milliseconds the source endpoint waits to retransmit message.	Yes

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>

Attribute	Description	Required?
`Milliseconds`	The number of milliseconds that defines a period of inactivity.	Yes

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>

Attribute	Description	Required?
`optional`	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`.	No

5

Web Service MakeConnection Policy Assertion Reference

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/`.
This chapter includes the following sections:

> **Note:**
>
> This section applies *only* to JAX-WS 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.
- 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.
- Element Descriptions

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 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.

wsp:Policy
    wsmc:MCSupported

- 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>

Attribute	Description	Required?
`optional`	Specifies whether MakeConnection must be used by the web service client. Valid values for this attribute are `true` and `false`. Default value is `true`. If set to `false`, both ReplyTo and FaultTo headers must contain MakeConnection anonymous URIs.	No

6

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-securitypolicy-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.

This chapter includes the following sections:

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

```
<wsp:Policy
  xmlns:wsp="http://schemas.xmlsoap.org/ws/2004/09/policy"
  xmlns:wssp="http://www.bea.com/wls90/security/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*.

See Configuring Message-Level Security in *Securing WebLogic Web Services for Oracle WebLogic Server* for task-oriented information about creating a message-level secured WebLogic web service.

Example of a Policy File With Security Elements

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

```
<?xml version="1.0"?>
<wsp:Policy
  xmlns:wsp="http://schemas.xmlsoap.org/ws/2004/09/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>
  <wssp:Confidentiality>
    <wssp:KeyWrappingAlgorithm
          URI="http://www.w3.org/2001/04/xmlenc#rsa-1_5"/>
    <wssp:Target>
      <wssp:EncryptionAlgorithm
          URI="http://www.w3.org/2001/04/xmlenc#tripledes-cbc"/>
```

```
      <wssp:MessageParts
           Dialect="http://www.bea.com/wls90/security/policy/wsee#part">
           wls:SecurityHeader(Assertion)
      </wssp:MessageParts>
    </wssp:Target>
    <wssp:Target>
      <wssp:EncryptionAlgorithm
           URI="http://www.w3.org/2001/04/xmlenc#tripledes-cbc"/>
      <wssp:MessageParts
           Dialect="http://schemas.xmlsoap.org/2002/12/wsse#part">
           wsp:Body()</wssp:MessageParts>
    </wssp:Target>
    <wssp:KeyInfo />
  </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
- Claims
- Confidentiality
- ConfirmationMethod
- DigestAlgorithm

- EncryptionAlgorithm
- Identity
- Integrity
- KeyInfo
- KeyWrappingAlgorithm
- Label
- Length
- MessageAge
- MessageParts
- Policy
- SecurityToken
- SecurityTokenReference
- SignatureAlgorithm
- SupportedTokens
- Target
- TokenLifeTime
- Transform
- UsePassword

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>

Attribute	Description	Required?
`URI`	The algorithm used to canonicalize the SOAP message being signed. You can specify only the following canonicalization algorithm: `http://www.w3.org/2001/10/xml-exc-c14n#`	Yes

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>

Attribute	Description	Required?
`SupportTrust10`	The valid values for this attribute are `true` and `false`. The default value is `false`.	No

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:

```
<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 version="1.0"?>
<wsp:Policy
  xmlns:wsp="http://schemas.xmlsoap.org/ws/2004/09/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 version="1.0"?>
<wsp:Policy
  xmlns:wsp="http://schemas.xmlsoap.org/ws/2004/09/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>
    <wssp:SignatureAlgorithm
       URI="http://www.w3.org/2000/09/xmldsig#rsa-sha1"/>
    <wssp:CanonicalizationAlgorithm
       URI="http://www.w3.org/2001/10/xml-exc-c14n#"/>
    <wssp:Target>
      <wssp:DigestAlgorithm
         URI="http://www.w3.org/2000/09/xmldsig#sha1" />
      <wssp:MessageParts
         Dialect="http://schemas.xmlsoap.org/2002/12/wsse#part">
         wsp:Body()
      </wssp:MessageParts>
    </wssp:Target>
    <wssp:SupportedTokens>
      <wssp:SecurityToken
          IncludeInMessage="true"
          TokenType="http://docs.oasis-open.org/wss/2004/01/oasis-2004-01-saml-token-
profile-1.0#SAMLAssertionID">
        <wssp:Claims>
          <wssp:ConfirmationMethod>holder-of-key</wssp:ConfirmationMethod>
        </wssp:Claims>
      </wssp:SecurityToken>
    </wssp:SupportedTokens>
  </wssp:Integrity>
</wsp:Policy>
```

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>

Attribute	Description	Required?
`URI`	The digest algorithm that is used when digitally signing the specified parts of a SOAP message. You can specify only the following digest algorithm: `http://www.w3.org/2000/09/xmldsig#sha1`	Yes

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>

Attribute	Description	Required?

`URI`	The encryption algorithm used to encrypt specified parts of the SOAP message.
	Valid values are:
	`http://www.w3.org/2001/04/xmlenc#tripledes-cbc`
	`http://www.w3.org/2001/04/xmlenc#kw-tripledes`
	`http://www.w3.org/2001/04/xmlenc#aes128-cbc`
	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.
		Yes

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>

Attribute	Description	Required?
`SignToken`	Specifies whether the security token, specified using the `<SecurityToken>` child element of `<Integrity>`, 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`.	No
`SupportTrust10`	The valid values for this attribute are `true` and `false`. The default value is `false`.	No
`X509AuthConditional`	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.	No

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>

Attribute	Description	Required?
`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-mgf1p` (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.

This element has no attributes.

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:wsp="http://schemas.xmlsoap.org/ws/2004/09/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="Dy2PFsX3ZQacqNKEANpXbNMnMhm2BmGOA2WDc2E0JpiaaTmbYNwT"
     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:Expires>2005-11-09T17:51:55Z</wsu:Expires>
</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 6-7 Attributes of <MessageAge>

Attribute	Description	Required?
`Age`	Specifies the actual maximum age time-out for a SOAP message, in seconds.	No

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

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

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>

Attribute	Description	Required?
`Dialect`	Identifies the dialect used to identity 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

Policy

Groups nested policy assertions.

SecurityToken

Specifies the security token that is supported for authentication, encryption or digital signatures, depending on the parent element.

For example, if this element is defined in the `<Identity>` parent element, then is specifies that a client application, when invoking the web service, must attach a security token to the SOAP request. For example, a web service might require that the client application present a SAML authorization token issued by a trusted authorization authority for the web service to be able to access sensitive data. If this element is part of `<Confidentiality>`, then it specifies the token used for encryption.

The specific type of the security token is determined by the value of its `TokenType` attribute, as well as its parent element.

By default, a security token for a secure conversation has a lifetime of 12 hours. To change this default value, add a `<Claims>` child element that itself has a `<TokenLifeTime>` child element, as described in Claims.

Table 6-10 Attributes of <SecurityToken>

Attribute	Description	Required?
`DerivedFromTokenType`	Specifies what security token it is derived from. For example, a value of "`http://schemas.xmlsoap.org/ws/2005/02/sc/sct`" specifies that it is derived from an old version of Secure Conversation Token.	No

Table 6-10 (Cont.) Attributes of <SecurityToken>

Attribute	Description	Required?
`IncludeInMessage`	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 `<Confidentiality>` assertion and `true` when used in the `<Integrity>` assertion. The value of this attribute is *always* `true` when used in the `<Identity>` assertion, even if you explicitly set it to `false`.	No
`TokenType`	Specifies the type of security token. Valid values are: • `http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-x509-token-profile-1.0#X509v3` (To specify a binary X.509 token) • `http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-username-token-profile-1.0#UsernameToken` (To specify a username token) • `http://docs.oasis-open.org/wss/2004/01/oasis-2004-01-saml-token-profile-1.0#SAMLAssertionID` (To specify a SAML token)	Yes

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>

Attribute	Description	Required?

`URI`	Specifies the cryptographic algorithm used to compute the signature.
	Note: Be sure that you specify an algorithm that is compatible with the certificates you are using in your enterprise.
	Valid values are:
	`http://www.w3.org/2000/09/xmldsig#rsa-sha1`
	`http://www.w3.org/2000/09/xmldsig#dsa-sha1`
		Yes

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>

Attribute	Description	Required?
`encryptContentOnly`	Specifies whether to encrypt an entire element, or just its content. This attribute can be specified only when `<Target>` is a child element of `<Confidentiality>`. Default value of this attribute is `true`, which means that only the content is encrypted.	No

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>

Attribute	Description	Required?
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>

Attribute	Description	Required?
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.

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:

```
<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:

```
<wssp:MessageParts
     Dialect="http://www.w3.org/TR/1999/REC-xpath-19991116">
        wsp:GetHeader(./wsse: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` 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:

```
<wssp:MessageParts
     Dialect="http://schemas.xmlsoap.org/2002/12/wsse#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:

- `wsrm:SequenceAcknowledgement`
- `wsrm:AckRequested`
- `wsrm: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:

```
<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:

```
<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.

7

WebLogic Web Service Deployment Descriptor Schema Reference

The WebLogic equivalent to the standard Jakarta 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.

This chapter includes the following sections:

Overview of weblogic-webservices.xml

The standard Jakarta 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://www.oracle.com/webfolder/technetwork/jsc/xml/ns/javaee/javaee_web_services_1_4.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 Jakarta 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 Jakarta 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 Jakarta 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 Remote 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 Jakarta EE Schema for the `web.xml` file. See the Servlet Deployment Descriptor Schema at `http://www.oracle.com/webfolder/technetwork/jsc/xml/ns/javaee/web-app_4_0.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 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.

```
<weblogic-webservices> {1}
   <webservice-description> {1 or more}
      <webservice-description-name> {1 or more}
      <webservice-type> {0 or 1}
```

```
<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}
                <jndi-connection-factory-name> {0 or 1}
                <jndi-initial-context-factory> {0 or 1}
                <jndi-url> {0 or 1}
                <jndi-context-parameter> {0 or 1}
                <time-to-live> {0 or 1}
                <priority> {0 or 1}
                <delivery-mode> {0 or 1}
                <reply-to-name> {0 or 1}
                <target-service> {0 or 1}
                <binding-version> {0 or 1}
                <message-type> {0 or 1}
                <enable-http-wsdl-access> {0 or 1}
                <run-as-principal> {0 or 1}
                <run-as-role> {0 or 1}
                <mdb-per-destination> {0 or 1}
                <activation-config> {0 or 1}
      <fastinfoset> {0 or 1}
      <logging-level> {0 or 1}
   <webservice-security> {0 or 1}
      <mbean-name> {1}
```

- 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
- logging-level
- login-config
- lookup-variant
- mbean-name
- mdb-per-destination
- message-type
- messaging-queue-jndi-name
- messaging-queue-mdb-run-as-principal-name
- name
- non-buffered-destination
- non-buffered-source
- operation
- persistence-config
- port-component
- port-component-name
- priority
- reliability-config
- reply-to-name
- request-queue
- response-queue
- retransmission-exponential-backoff
- retry-count
- retry-delay
- run-as-principal
- run-as-role
- sequence-expiration
- service-endpoint-address
- soapjms-service-endpoint-address
- stream-attachments
- target-service
- time-to-live
- transport-guarantee

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:

`PnYnMnDTnHnMS`

Table 7-1 describes the duration format fields. This value defaults to `P0DT3S` (3 seconds).

Table 7-1 Duration Format Description

Field	Description
*n*Y	Number of years (*n*).
*n*M	Number of months (*n*).
*n*D	Number of days (*n*).
T	Date and time separator.
*n*H	Number of hours (*n*).
*n*M	Number of minutes (*n*).
*n*S	Number of seconds (*n*).

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://www.oracle.com/webfolder/technetwork/jsc/xml/ns/javaee/web-app_4_0.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:

`PnYnMnDTnHnMnS`

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;...;name`*`N`*`=value`*`N`*.

JNDI properties. Each property is specified using name-value pairs, separated by semicolons (;). For example: `name1=value1;...;name`*`N`*`=value`*`N`*.

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-connection-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://www.oracle.com/webfolder/technetwork/jsc/xml/ns/javaee/web-app_4_0.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 Remote 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:

`PnYnMnDTnHnMnS`

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 `<run-as-principal>` child element of the `<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 `<run-as-role>` child element of the `<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 `<sequence-expiration>` child element of the `<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:

`PnYnMnDTnHnMnS`

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 `<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 `<webservice-contextpath>` and `<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://www.oracle.com/webfolder/technetwork/jsc/xml/ns/javaee/web-app_4_0.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:

PnYnMnDTnHnMnS

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:

PnYnMnDTnHnMnS

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 or the `<WLHttpTransport>` child element of `jwsc`.

webservice-type

The `<webservice-type>` element specifies the web service based on the JAX-WS standard. Valid values is `JAXWS`.

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.

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.