BEA WebLogic Workshop™ Help

Version 8.1 SP4
December 2004

Copyright

Restricted Rights Legend

Trademarks or Service Marks

Table of Contents

Configuration File Reference

The following configuration files are associated with WebLogic Workshop.

Topics Included in This Section

jws−config.properties Configuration File

The jws−config.properties file specifies WebLogic Workshop runtime configuration for a WebLogic Server domain.

workshopLogCfg.xml Configuration File

The workshopLogCfg.xml file specifies the WebLogic Workshop logging configuration and logging levels for WebLogic Server. By default this configuration file writes debugging information to the log file weblogic_debug.log. (Use this log file when working with technical support.) By default, exceptions are written to workshop_errors.log. Both weblogic_debug.log and weblogic_errors.log are located in the BEA_HOME\weblogic81\workshop directory.

wlw−config.xml Configuration File

The wlw−config.xml file allows you to configure runtime parameters of web services, both on the development server and production servers. wlw−config.xml is specific to a WebLogic Workshop application. Note that values entered in the wlw−config.xml file cannot be overridden by other runtime configuration mechanisms, because these values become hardcoded into the deployment EAR file. For this reason, we recommend that you use the wlw−runtime−config.xml file for most runtime configuration purposes.

wlw−runtime−config.xml Configuration File

The wlw−runtime−config.xml file allows you to configure runtime parameters of web services, both on the development server and production servers. wlw−runtime−config.xml is specific to a WebLogic Workshop application.

wlw−manifiest.xml Configuration File

The wlw−manifest.xml file is autogenerated when a WebLogic Workshop application is compiled into a deployable EAR file. This file tells server administrators what resources are necessary on the EAR file's target server.

Web Service Security Policy Configuration File (WSSE File)

WSSE files configure security for web services that implement the web serivces security standard.

jws−config.properties Configuration File

The jws−config.properties file specifies domain−wide configuration parameters for the WebLogic Workshop runtime. The file is located in the domain root directory.

Properties

The jws−config.properties file defines the following properties:

weblogic.jws.InternalJMSServer=<JMS server name>

Specifies the name of the JMS server to be used by WebLogic Workshop JMS controls, JMS transport and JMS queues used as message buffers. The named JMS server must be configured in the current domain.

Default value: cgJMSServer

weblogic.jws.InternalJMSConnFactory=<JMS connection factory JNDI name>

Specifies the JNDI name of the JMS connection factory to be used by WebLogic Workshop JMS controls, JMS transport and JMS queues used as message buffers. The named JMS connection factory must be configured in the current domain.

Default value: weblogic.jws.jms.QueueConnectionFactory

weblogic.jws.ConversationDataSource=<data source JNDI name>

Specifies the JNDI name of the data source to be used for conversational persistence. The named data source must be configured in the current domain.

Default value: cgDataSource

weblogic.jws.JMSControlDataSource=<data source JNDI name>

Specifies the JNDI name of the data source to be used for JMS control state management. The named data source must be configured in the current domain.

Default value: cgDataSource

weblogic.jws.ConversationMaxKeyLength=<integer>

Specifies the maximum length of a conversation ID. Conversational web services have an associated conversation identifier. The identifier is proposed by the client and must be guaranteed to be unique across all web services and conversations in the current domain.

Default value: 768

Note: The conversation ID is used as the primary key in a conversation state table that resides in the database associated with the data source identified by weblogic.jws.ConversationDataSource (above). Setting weblogic.jws.ConversationMaxKeyLength to a value larger than what can be accommodated by the associated database table will result in errors.

weblogic.jws.MessageTransactionTimeout=<integer>

The timeout for asynchronous message requests. Takes an integer indicating seconds.

Related Topics

How Do I: WebLogic Workshop−Enable an Existing WebLogic Server Domain?

How Do I: Create a New WebLogic Workshop−enabled WebLogic Server Domain?

workshopLogCfg.xml Configuration File

The workshopLogCfg.xml file specifies the WebLogic Workshop logging configuration and logging levels for WebLogic Server. You may use this logging facility in your applications. You may also be directed by BEA Technical Support personnel to modify WebLogic Workshop's logging behavior in order to diagnose problems you experience. This topic describes the basic features of WebLogic Workshop's logging mechanism.

Log4j Library

WebLogic Workshop uses the Log4j Java logging facility developed by the Jakarta Project of the Apache Foundation. The names of the XML tags in workshopLogCfg.xml directly correspond to names of classes and fields defined in the Log4j API Specification. You can learn more about Log4j at The Log4j Project. A brief introduction to Log4j is included below.

Loggers

Log4j defines the Logger class. An application may create multiple loggers, each with a unique name. In a typical usage of Log4j, an application creates a Logger instance for each application class that will emit log messages. The name of the logger is typically the same as the partially qualified name of the application class. For example, the application class com.mycompany.MyClass might create a Logger with the name "mycompany.MyClass". Loggers exist in a namespace hierarchy and inherit behavior from their ancestors in the hierarchy.

The Logger class defines four methods for emitting log messages: debug, info, warn and error. The application class invokes the appropriate method (on its local Logger) for the situation being reported.

Appenders

Log4j defines Appenders to represent destinations for logging output. Multiple Appenders may be defined. For example, an application may define an Appender that sends log messages to the console, and another Appender that writes log messages to a file. Individual Loggers may be configured to write to zero or more Appenders. One example usage would be to send all logging messages (all levels) to a log file, but only error level messages to the console.

Layouts

Log4J defines Layouts to control the format of log messages. Each Layout specifies a particular message format in which may be substituted the data such as the current time and date, the log level, the Logger name, the log message and other information. A specific Layout is associated with each Appender. This allows you to specify a different log message format for console output than for file output, for example.

Configuration

All aspects of Log4j configuration at runtime. This is typically accomplished with an XML configuration file whose format is defines by the Log4j library. The configuration file may be specified at runtime using the log4j.configuration Java property.

The configuration file may declare Loggers, Appenders and Layouts and configure combinations of them to provide the logging style desired by the application.

WebLogic Workshop Logging

By default, WebLogic Workshop's logging configuration is defined in the file <BEA_HOME>/<WEBLOGIC_HOME>/common/lib/workshopLogCfg.xml. The workshopLogCfg.xml file contains XML tags that describe how the WebLogic Workshop environment logs various types of messages. The names of these tags directly correspond to names of classes and fields defined in the Log4j API Specification.

If you want to customize the way WebLogic Workshop logs messages, you can either modify workshopLogCfg.xml, or provide your own Log4j configuration file. If you provide your own version, you need to set the log4j.configuration Java property to the location of this file. For example, using the same command line that you used to start WebLogic Server, type −Dlog4j.configuration=<path to config file>.

It is helpful to understand the contents of the workshopLogCfg.xml. Once you understand the elements and attributes defined in this file, you can either use this knowledge to customize workshopLogCfg.xml, or write a separate Log4j configuration file that better suits your logging requirements.

Understanding the Contents of workshopLogCfg.xml

The workshopLogCfg.xml file contains two major sets of configuration values. The first set of configurations use the <appender> element to define log files that receive messages from the WebLogic Workshop environment. Note that this element shares the same name as the Appender class defined in the Log4j API Specification. The following snippet illustrates an entry in this section of the configuration file:

```
<appender name="SYSLOGFILE" class="org.apache.log4j.RollingFileAppender">
  <param name="File"   value="workshop_debug.log" />
  <param name="Append" value="true" />
  <param name="MaxFileSize" value="500000KB" />
  <layout class="org.apache.log4j.PatternLayout">
    <param name="ConversionPattern" value="%d{DATE} %-5p %-15c{1} [%t][%x]: %m%n"/>
  </layout>
</appender>
```

This entry causes WebLogic Workshop to create a Log4j Appender object of type RollingFileAppender. RollingFileAppender is a special kind of Log4j Appender class that backs up log files when they reach a certain size. The Log4j specification defines several different types of Appender classes. You can use the class attribute of the <appender> element to use any of these types. For instance, if you would like WebLogic Workshop to send certain types of messages to the console, then you can set the class attribute of an <appender> element to org.apache.log4j.ConsoleAppender. The name attribute of the <appender> element assigns a variable name to the Appender object. workshopLogCfg.xml uses that name later in the configuration file to assign specific messages to that Appender.

The <param> elements use name−value pairs to assign values to various fields of the Appender object. The File value defines the name of the physical file to which message sources log their messages. To learn more about the default log files to which WebLogic Workshop writes log messages, see Message Logging. The Append value indicates that WebLogic Workshop should append any new log messages to the end of the log file. The MaxFileSize value specifies the file size limit of this log file.

Finally the class attribute of the layout element defines a Log4j Layout class. Layout classes define the format of log messages. The ConversionPattern value further defines these formats. For more information on providing conversion patterns, see the Log4j API Specification.

The second set of configurations use the <category> element to define message sources. It also relates these sources to log files defined in the <append> elements of the workshopLogCfg.xml. The following snippet illustrates an entry in this portion of the configuration file:

```
<category name="weblogic.servlet.jsp">
    <priority value="warn" />
    <appender−ref ref="SYSLOGFILE" />
    <appender−ref ref="SYSERRORLOGFILE" />
  </category>
```

Note that the ref attribute of the <appender−ref> element refers to the names of Appender objects defined in the first half of workshopLogCfg.xml. The priority value defines the lowest priority message type that WebLogic Workshop will send to these Appender objects. In this case, weblogic.servlet.jsp reports all messages of type warn or higher. This means that weblogic.servlet.jsp will also log messages of type error since messages of type error have a higher priority than messages of type warn. Info, warn, and error are all Log4j message types. For more information, see the Log4j Specification.

The name attribute of the <category> element defines the message source. In this case, WebLogic Workshop sends all warning messages originating from a JSP file to both the SYSLOGFILE and SYSERRORLOGFILE Appender objects.

The workshopLogCfg.xml file also defines three category elements that are useful when debugging the request−response chain associated with a JWS (WebLogic Workshop web service) invocation. The categories are defined in the configuration file as shown here:

```
  <category name="WLW.REQUEST">
      <priority value="info"/>
      <appender−ref ref="APPLOGFILE"/>
  </category>

  <category name="WLW.RESPONSE">
      <priority value="info"/>
      <appender−ref ref="APPLOGFILE"/>
  </category>

  <category name="WLW.INVOKE">
      <priority value="warn"/>
      <appender−ref ref="APPLOGFILE"/>
  </category>
```

You may adjust the priority levels of these categories to enable or disable logging for the entire call stack that is involved in web service method invocation.

Adding Log4J Log Messages To Your Application Code

To cause your code to emit messages to the workshop.log file, obtain a Logger for your file. You may obtain a Logger by calling the JwsContext.getLogger method, typically with the qualified name of your class as the Logger name (e.g. "async.HelloWorldAsync" for the sample web service async/HelloWorldAsync.jws). Then simply call the Logger's debug, info, warn or error methods as appropriate.

The default logging level is info. This means log messages emitted by Logger.debug() will not be logged by default. To enable logging of all message levels, set the log level for the "wlw" category to debug in workshopLogCfg.xml, as shown here:

```
  <!-- The wlw category is used for messages logged using the Logger API from JwsContext and Co
  <category name="wlw">
    <priority value="debug" />
    <appender-ref ref="APPLOGFILE" />
  </category>
```

The example code below illustrates use of logging in a JWS file. Portions of the source file have been omitted for brevity.

```
package async;
import com.bea.control.JwsContext;
import com.bea.control.TimerControl;
import com.bea.wlw.util.Logger;

public class HelloWorldAsync
{
    /** @common:context */
    JwsContext context;
    /**
     * @common:operation
     * @jws:conversation phase="start"
     */
    public void HelloAsync()
    {
        Logger logger = context.getLogger("async.HelloWorldAsync");
        logger.debug("about to start timer");
        // all we do here is start the timer.
        helloDelay.start();
        logger.debug("timer started");
        return;
    }
    private void helloDelay_onTimeout(long time)
    {
        Logger logger = context.getLogger("async.HelloWorldAsync");
        // send the client a hello message.
        logger.debug("in timer handler: calling client");
        callback.onHelloResult("Hello, asynchronous world");

        // we don't want any more timer events for this conversation.
        logger.debug("in timer handler: stopping timer");
        helloDelay.stop();

        return;
    }
}
```

The code above results in the following content in workshop.log when the HelloAsync method of HelloWorldAsync.jws is invoked. Log entries not emitted by the preceding code have been removed from the output below.

```
13 Jun 2003 14:49:23,796 DEBUG HelloWorldAsync: about to start timer
13 Jun 2003 14:49:24,171 DEBUG HelloWorldAsync: timer started
13 Jun 2003 14:49:33,921 DEBUG HelloWorldAsync: in timer handler: calling client
13 Jun 2003 14:49:33,937 DEBUG HelloWorldAsync: in timer handler: stopping timer
```

Tracking Log Information in a Clustered Environment

If you deploy your application to a clustered environment, you may want to track log information for individual servers in the cluster. You can modify the configuration for Appenders in workshopLogCfg.xml to include a variable that returns the name of the server which logged the message. When WebLogic Server starts, it sets an environment variable weblogic.Name with the server name. You can refer to this variable from the log file.

If the number of log messages you expect is fairly low and there are not too many servers in the cluster writing to a single log file all at once, you can modify each Appender so that the server name is written out for each logged message. In this case you should modify the value of the <param> tag named ConversionPattern, which is a child of the <layout> tag, to include the server variable. Here's an example of what the modified Appender would look like:

```
<appender name="APPLOGFILE" class="org.apache.log4j.RollingFileAppender">
  <param name="File" value="workshop.log" />
  <param name="Append" value="true" />
  <param name="MaxFileSize" value="3000KB" />
  <layout class="org.apache.log4j.PatternLayout">
    <param name="ConversionPattern" value="server:${weblogic.Name} %d{DATE} %-5p %-15c{1}: %m%n
  </layout>
</appender>
```

Note that the value of the ConversionPattern parameter includes the variable ${weblogic.Name}. This pattern now writes an entry like the following example to the log file when a message is logged. Note that the name of the server appears as cgServer.

```
server:cgServer 05 Jun 2003 10:50:21,194 INFO RMClient: ConversationID=1054835390772; Protocol=
URI=/R MClientWeb/TestDriver.jws;Method=onTimeout; Phase=continue;Callback=null
```

If you are expecting a high volume of log messages, you can modify each Appender so that a separate log file is written for each server. In this case you modify the value of the <param> tag named File to prepend the server name to generate a unique log file for each server, as shown in the following example:

```
<appender name="APPLOGFILE" class="org.apache.log4j.RollingFileAppender">
  <param name="File" value="${weblogic.Name}.workshop.log" />
  <param name="Append" value="true" />
  <param name="MaxFileSize" value="3000KB" />
  <layout class="org.apache.log4j.PatternLayout">
    <param name="ConversionPattern" value="%d{DATE} %-5p %-15c{1}: %m%n"/>
  </layout>
</appender>
```

Related Topics

Message Logging

Configuration File Reference

wlw−config.xml Configuration File

The wlw−config.xml file allows you to configure runtime parameters of web services, both on the development server and production servers. This file is used by wlwBuild when preparing WebLogic Workshop application EAR files for deployment to a production server.

Note that the values appearing in wlw−config.xml are hardcoded into the deployment EAR and cannot be overridden by other runtime configuration mechanisms. For most cases you should use wlw−runtime−config.xml to configure the runtime information for your Workshop application.

General Structure

```
<wlw−config>
   <hostname>
   <protocol>
   <http−port>
   <https−port>
   <transaction−isolation−level>
   <transaction−timeout>
   <message−transaction−timeout>
   <ejb−concurrency−strategy>
   <max−beans−in−cache>
   <idle−timeout−seconds>
   <read−timeout−seconds>
   <initial−beans−in−free−pool>
   <max−beans−in−free−pool>
   <web−tier−controls>
      <class−name>
   <component−group URI="" protocol="">
      <component URI="" default=boolean>
   <service>
      <class−name>
      <protocol>
```

<wlw−config>

The <wlw−config> element is the root element of the wlw−config.xml file.

```
<wlw−config>
   <hostname>
   <protocol>
   <http−port>
   <https−port>
   <transaction−isolation−level>
   <transaction−timeout>
   <message−transaction−timeout>
   <ejb−concurrency−strategy>
   <max−beans−in−cache>
   <idle−timeout−seconds>
```

<read−timeout−seconds>
<initial−beans−in−free−pool>
<max−beans−in−free−pool>
<web−tier−controls>
<component−group URI="" protocol="">
<service>

Syntax

```
<wlw−config>
<!−−
Children elements defining the runtime information about the deployed application.
−−>
</wlw−config>
```

Attributes

none.

Hierarchy

Parents: none.

Children: <hostname>, <protocol>, <http−port>, <https−port>, <transaction−isolation−level>, <transaction−timeout>, <message−transaction−timeout>, <ejb−concurrency−strategy>, <max−beans−in−cache>, <idle−timeout−seconds>, <read−timeout−seconds>, <initial−beans−in−free−pool>, <max−beans−in−free−pool>, <component−group>, <service>.

<hostname>

The <hostname> element specifies the name of the machine where your application will be deployed. If the value is not set in wlw−config.xml, the value may also be set at runtime in the wlw−runtime−config.xml file, which resides in the server's domain root folder. Also, it may be set at runtime via the server configuration parameter FrontendHost: in the WebLogic Server console navigate to Server −−> Protocols −−> HTTP −−> Advanced Options.

Note: that if you specify the <hostname> in wlw−config.xml, the value will be fixed and it *cannot* be overridden by other configuration techniques. For this reason it is generally preferable to use the alternative configuation techniques described above.

<wlw−config>
<hostname>

Syntax

```
<hostname> hostnameString </hostname>
```

Attributes

none.

Hierarchy

Parents: <wlw−config>.

Children: none.

<protocol>

The <protocol> element defines the default exposure protocol for your web application. Unless otherwise specified in the <service> element, web resources within the web application will be exposed on the protocol specified here. Possible values are http or https.

<wlw−config>
  <protocol>

Syntax

```
<protocol> [HTTP | HTTPS] </protocol>
```

Attributes

none.

Hierarchy

Parents: <wlw−config>.

Children: none.

<http−port>

The <http−port> element specifies which port should be used for HTTP traffic. The range of values is 1 to 65535. If omitted the value is set to 7001.

<wlw−config>
  <http−port>

Syntax

```
<http−port> httpPort </http−port>
```

Attributes

none.

Hierarchy

Parents: <wlw−config>.

Children: none.

<https−port>

The <https−port> element specifies which port should be used for https traffic. The range of values is 1 to 65535. If omitted the value is set to 7002.

<wlw−config>
  <https−port>

Syntax

```
<https−port> httpsPort </https−port>
```

Attributes

none.

Hierarchy

Parents: <wlw−config>.

Children: none.

<transaction−isolation−level>

The <transaction−isolation−level> tag specifies the transaction isolation level of the EJB that is produced when a web service is compiled. The value of <transaction−isolation−level> will be mapped to the EJB's deployment descriptors at compile time. If no value is specified, then no value is written to the deployment descriptor; this will cause the runtime container to use whichever isolation level the target database (used by the EJB) is configured for. Valid values are

TRANSACTION_READ_COMMITTED

TRANSACTION_READ_UNCOMMITTED

TRANSACTION_REPEATABLE_READ

TRANSACTION_SERIALIZABLE

<wlw−config>
  <transaction−isolation−level>

Syntax

```
<transaction-isolation-level>
  [ TRANSACTION_READ_COMMITTED | TRANSACTION_READ_UNCOMMITTED | TRANSACTION_REPEATABLE_READ | T
</transaction-isolation-level>
```

Attributes

none.

Hierarchy

Parents: <wlw−config>.

Children: none.

<transaction−timeout>

The transaction−timeout element specifies the maximum duration for an EJB's container−initiated transactions. If a transaction lasts longer than transaction−timeout, WebLogic Server rolls back the transaction. Note that this element applies to Workshop web services, since web services are compiled into transactional EJBs. Takes an integer indicating seconds. The default value is 30 seconds.

<wlw−config>
  <transaction−timeout>

Syntax

```
<transaction-timeout> timoutSeconds </transaction-timeout>
```

Attributes

none.

Hierarchy

Parents: <wlw−config>.

Children: none.

<message−transaction−timeout>

The message−transaction−timeout element specifies the maximum duration for asynchronous message requests. If a transaction lasts longer than message−transaction−timeout, WebLogic Server rolls back the transaction. Takes an integer indicating seconds. The default value is 30 seconds.

<wlw−config>
  <message−transaction−timeout>

Syntax

```
<message−transaction−timeout> timoutSeconds </message−transaction−timeout>
```

Attributes

none.

Hierarchy

Parents: <wlw−config>.

Children: none.

<ejb−concurrency−strategy>

The <ejb−concurrency−strategy> element is used to specify the EJB concurrency strategy of the EJBs that back conversational web services. Valid values are

Exclusive

Database

Optimistic

Default value is Exclusive for non−production builds, and Database for production builds. For clustered environments a value of Database is required. Non−clustered environments may use Exclusive.

WebLogic Workshop requires that method invocations on conversational web services occur serially. Serial access is enforced via record locking on the conversation's database record. When SQL Server or Oracle is used as the persistent store for conversation state, record locking is provided by the SQL statements used internally and <ejb−concurrency−strategy> may be set to Optimistic. When PointBase is used as the conversation store, <ejb−concurrency−strategy> must be Exclusive or race conditions may occur in conversation state access.

Note: if your project contains stateful processes, compilation will fail if you set the <ejb−concurrency−strategy> value to "optimistic". Some examples of stateful processes are: (1) an asynchronous web service or (2) a business process that involves stateful nodes or transactions. For more information on stateful web services and business processes see Default Transactional Behavior in WebLogic Workshop and Building Stateless and Stateful Business Processes

<wlw−config>
  <ejb−concurrency−strategy>

Syntax

```
<ejb−concurrency−strategy> [ Exclusive | Database | Optimistic ] </ejb−concurrency−strategy>
```

Attributes

none.

Hierarchy

Parents: <wlw−config>.

Children: none.

<max−beans−in−cache>

The max−beans−in−cache element specifies the maximum number of objects of this project that are allowed in memory. When max−bean−in−cache is reached, WebLogic Server passivates some EJBs that have not recently been used by a client. max−beans−in−cache also affects when EJBs are removed from the WebLogic Server cache, as described in Stateful Session EJB Life Cycle. The default value is 1000.

<wlw−config>
  <max−beans−in−cache>

Syntax

```
<max−beans−in−cache> maxInteger </max−beans−in−cache>
```

Attributes

none.

Hierarchy

Parents: <wlw−config>.

Children: none.

<idle−timeout−seconds>

idle−timeout−seconds defines the maximum length of time a stateful EJB should remain in the cache. After this time has elapsed, WebLogic Server removes the bean instance if the number of beans in cache approaches the limit of max−beans−in−cache. The removed bean instances are passivated. See Stateful Session EJB Life Cycle for more information. The default value is 600 seconds.

<wlw−config>
  <idle−timeout−seconds>

Syntax

```
<idle−timeout−seconds> timeoutSeconds </idle−timeout−seconds>
```

Attributes

none.

Hierarchy

Parents: <wlw−config>.

Children: none.

<read−timeout−seconds>

The read−timeout−seconds element specifies the number of seconds between ejbLoad() calls on a Read−Only entity bean. By default, read−timeout−seconds is set to 600, and WebLogic Server calls ejbLoad() only when the bean is brought into the cache.

<wlw−config>
  <read−timeout−seconds>

Syntax

```
<read−timeout−seconds> secondsInteger </read−timeout−seconds>
```

Attributes

none.

Hierarchy

Parents: <wlw−config>.

Children: none.

<initial−beans−in−free−pool>

If you specify a value for initial−beans−in−free−pool, you set the initial size of the pool. WebLogic Server populates the free pool with the specified number of bean instances for every bean class at startup. Populating the free pool in this way improves initial response time for the EJB, because initial requests for the bean can be satisfied without generating a new instance. Default value is 0.

<wlw−config>
  <initial−beans−in−free−pool>

Syntax

```
<initial−beans−in−free−pool> beansInteger </initial−beans−in−free−pool>
```

Attributes

none.

Hierarchy

Parents: <wlw−config>.

Children: none.

<max−beans−in−free−pool>

WebLogic Server maintains a free pool of EJBs for every stateless session bean and message−driven bean class. The max−beans−in−free−pool element defines the size of this pool. By default, max−beans−in−free−pool has no limit; the maximum number of beans in the free pool is limited only by the available memory. See Stateful Session EJB Life Cycle for more information. Default value is 1000.

<wlw−config>
  <max−beans−in−free−pool>

Syntax

```
<max−beans−in−free−pool> maxBeansInteger </max−beans−in−free−pool>
```

Attributes

none.

Hierarchy

Parents: <wlw−config>.

Children: none.

<web−tier−controls>

Controls code generation for controls referenced from the web tier. Use this element as an optimization to reduce the number of generated EJBs. If the <web−tier−controls> element is empty, then no EJBs will be generated. Note that referenced controls will successfully compile, but will fail at runtime.

<wlw−config>
  <web−tier−controls>
    <class−name>

Syntax

```
<web−tier−controls>
    <!−−
    Child elements listing control classes referenced from the web−tier
    −−>
```

```
</web-tier-controls>
```

Attributes

none.

Hierarchy

Parents: <wlw−config>.

Children: <class−name>.

<class−name>

Control file class names that are referenced from the web−tier. This element gives the fully qualified class name, including package. No file extension should be included. E.g., <class−name>my.package.name.Foo</class−name>.

<wlw−config>
  <web−tier−controls>
    <class−name>

Syntax

```
<class-name> JavaClassNameString </class-name>
```

Attributes

none.

Hierarchy

Parents: <web−tier−controls>.

Children: none.

<component−group>

The <component−group> element is used to group together different versions of a component.

<wlw−config>
  <component−group>
    <component>

Syntax

```
<component-group>
<!--
Child elements listing the different versions of the component
-->
```

```
</component-group>
```

Attributes

Attribute	Description
URI	Required string. Any string indicating part of the URI space.
protocol	Optional string.

Hierarchy

Parents: <wlw-config>.

Children: <component>.

Example

```
 <component-group URI="version/MyWebService.jws">
    <component URI="version/MyWebService3.jws" default="true"/>
    <component URI="version/MyWebService2.jws" />
    <component URI="version/MyWebService1.jws" />
 </component-group>
```

<component>

Defines an individual version in a component group.

<wlw-config>
  <component-group>
    <component>

Syntax

```
<component
    URI="[uriString]"
    default="[boolean]"
/>
```

Attributes

Attribute	Description
URI	Required string.
default	Optional boolean.

Hierarchy

Parents: <component-group>.

Children: none.

Example

```
 <component-group URI="version/MyWebService.jws">
    <component URI="version/MyWebService3.jws" default="true"/>
    <component URI="version/MyWebService2.jws" />
    <component URI="version/MyWebService1.jws" />
 </component-group>
```

<service>

Defines a web service−protocol mapping. Use the child elements <class−name> and <protocol> elements to specify the transport protocol for individual web resources in a web application.

<wlw−config>
  <service>
    <class−name>
    <protocol>

Syntax

```
<service>
<!--
Child elements specifying the transport protocol for individual web resources.
-->
</service>
```

Attributes

none.

Hierarchy

Parents: <wlw−config>.

Children: <class−name>, <protocol>.

Example

```
<service>
    <class-name>HelloWorld</class-name>
    <protocol>http</protocol>
</service>
<service>
    <class-name>HelloWorldSecure</class-name>
    <protocol>https</protocol>
</service>
```

<class−name>

The <class−name> element is used in conjunction with the <service> and <protocol> elements to specify the transport protocols of individual web resources. Possible values are the class names of web resources. For example, to specify a web service Foo.jws, use <class−name>Foo</class−name>. Values of the

<class−name> element are case sensitive.

<wlw−config>
  <service>
    <class−name>

Syntax

```
<class−name> JavaClassString </class−name>
```

Attributes

none.

Hierarchy

Parents: <service>.

Children: none.

<protocol>

Defines an HTML protocol to be used by a given class / web resource. Valid values are either HTTP or HTTPS.

Specifies

<wlw−config>
  <service>
    <protocol>

Syntax

```
<protocol> protocolString </protocol>
```

Attributes

none.

Hierarchy

Parents: <service>.

Children: none.

Related Topics

wlwBuild Command

wlw−runtime−config.xml Configuration file

wlw−runtime−config.xml Configuration File

The wlw−runtime−config.xml file allows you to configure runtime parameters of web resources on a production server.

The wlw−runtime−config.xml file must be placed in the root of the WebLogic Server domain directory in order to impact the configuration of WebLogic Workshop applications deployed within the domain.

You can use the example at the bottom of this topic as a template wlw−runtime−config.xml file.

General Structure

```
<wlw−runtime−config>
  <wlw−config>
    <hostname>
    <protocol>
    <http−port>
    <https−port>
    <component−group>
      <component>
    <service>
      <class−name>
      <protocol>
```

<wlw−runtime−config>

The <wlw−runtime−config> element is the root element of the wlw−runtime−config.xml file. All other elements are children of <wlw−runtime−config>.

```
<wlw−runtime−config>
  <wlw−config>
```

Syntax

```
<wlw-runtime-config
    xmlns="stringNamespace"
>
```

Attributes

none.

Hierarchy

Parents: none.

Children: <wlw−config>.

<wlw−config>

The <wlw−config> element defines name and the context−path for a given deployed application. Include a <wlw−config> element for each deployed application. Child elements configure the application's hostname, transport protocol, etc.

<wlw−runtime−config>
  <wlw−config>
    <hostname>
    <protocol>
    <http−port>
    <https−port>
    <service>

Syntax

```
<wlw−config
    application−name="stringAppName"
    context−path="stringContextPath"
    >
<!−−
Child elements configure the application's hostname, transport protocol, etc.
−−>
</wlw−config>
```

Attributes

Attribute	Description
application−name	Required string. Specifies the name of the application.
context−path	Required string. The context−path attribute specifies the first part of the URL path, after the hostname. The following URL shows the location of the context path. http://localhost:7001/WebApp/binaryFlow/BinaryFlowController.jpf

Hierarchy

Parents: <wlw−runtime−config>.

Children: <hostname>, <protocol>, <http−port>, <https−port>, <service>.

<hostname>

Specifies the name of the machine where your web application will be deployed.

<wlw−runtime−config>
  <wlw−config>
    <hostname>

Syntax

```
<hostname> stringHostname </hostname>
```

Attributes

none.

Hierarchy

Parents: <wlw−config>.

Children: none.

<protocol>

Defines the default exposure protocol for your web application. Unless otherwise specified in the class−specific <protocol> element, web resources within the web application will be exposed on the protocol specified here. Possible values are http or https.

```
<wlw−runtime−config>
  <wlw−config>
    <protocol>
```

Syntax

```
<protocol> [ HTTP | HTTPS ] </protocol>
```

Attributes

none.

Hierarchy

Parents: <wlw−config>.

Children: none.

<http−port>

Specifies which port should be used for http traffic. The range of values is 1 to 65535. If omitted the value is set to 7001.

```
<wlw−runtime−config>
  <wlw−config>
    <http−port>
```

Syntax

```
<http-port> portInteger </http-port>
```

Attributes

none.

Hierarchy

Parents: <wlw−config>.

Children: none.

<https−port>

Specifies which port should be used for https traffic. The range of values is 1 to 65535. If omitted the value is set to 7002.

```
<wlw−runtime−config>
  <wlw−config>
    <https−port>
```

Syntax

```
<https-port> portInteger </https-port>
```

Attributes

none.

Hierarchy

Parents: <wlw−config>.

Children: none.

<service>

Use the <service>, and its children <class−name> and <protocol> elements, to specify the transport protocol for individual web resources in a web application.

```
<wlw−runtime−config>
  <wlw−config>
    <service>
      <class−name>
      <protocol>
```

Syntax

```
<service>
<!--
Child elements specifying a resource and exposure protocol.
-->
</service>
```

Attributes

none.

Hierarchy

Parents: <wlw−config> .

Children: <class−name>, <protocol>.

<class−name>

Specifies the transport protocols of individual web resources. Possible values are the class names of web resources. For example, to specify a web service Foo.jws, use <class−name>Foo</class−name>. Values of the <class−name> element are case sensitive.

<wlw−runtime−config>
  <wlw−config>
    <service>
      <class−name>

Syntax

```
<class−name> stringClassName </class−name>
```

Attributes

none.

Hierarchy

Parents: <service>.

Children: none.

<protocol>

Specifies the exposure protocol for individual web resources. Overrides the application−scoped <protocol> element.

<wlw−runtime−config>
  <wlw−config>

<service>
<protocol>

Syntax

```
<protocol> [ HTTP | HTTPS ] </protocol>
```

Attributes

none.

Hierarchy

Parents: <service>.

Children: none.

Example

```
<?xml version="1.0" encoding="UTF-8"?>
<wlw-runtime-config xmlns="http://www.bea.com/2003/03/wlw/config/">
  <wlw-config application-name="cluster_client" context-path="/cluster_clusterAsClientWeb">
    <hostname>myMachine</hostname>
    <protocol>http</protocol>
    <http-port>7120</http-port>
  </wlw-config>
</wlw-runtime-config>
```

Related Topics

wlwBuild Command

wlw-config.xml Configuration File

wlw−manifest.xml Configuration File

The wlw−manifest.xml file provides infomation about the server resources referenced in an EAR file built with the wlwBuild command. Server administrators should examine the wlw−manifest.xml file to determine the resources necessary for successful deployment. For detailed information about using the wlw−manifest.xml file in deployment, see How Do I: Deploy a WebLogic Workshop Application to a Production Server?

General Structure

```
<wlw−manifest>
  <project>
    <async−request−queue>
    <async−request−error−queue>
    <top−level−component>
      <conversation−state−table>
        <column>
          <column−name>
          <column−type>
      <external−callbacks>
        <control>
          <role−name>
      <roles−allowed>
        <role−name>
      <roles−referenced>
        <role−name>
```

<wlw−manifest>

The root level element of the wlw−manifest.xml file.

```
<wlw−manifest>
  <project>
```

Syntax

```
<wlw−manifest>
<!−−
Child elements specifying the individual projects within the application EAR.
−−>
</wlw−manifest>
```

Attributes

none.

Hierarchy

Parents: none.

Children: <project>.

<project>

Defines an individual project within the EAR file.

<wlw−manifest>
  <project>
    <async−request−queue>
    <async−request−error−queue>
    <top−level−component>

Syntax

```
<project
    name="stringName"
    >
<!−−
Child elements defining the JMS queues and EJBs (the compiled products of web services) within
−−>
</project>
```

Attributes

Attribute	Description
name	Required string.

Hierarchy

Parents: <wlw−manifest>.

Children: <async−request−queue>, <async−request−error−queue>, <top−level−component>.

<async−request−queue>

Defines the JMS queue where asynchronous requests to web services and business preocesses are queued.

<wlw−manifest>
  <project>
    <async−request−queue>

Syntax

```
<async−request−queue> queueNameString </async−request−queue>
```

Attributes

none.

Hierarchy

Parents: <project>.

Children: none.

<async−request−error−queue>

Defines the JMS queue where errors associated with asynchronous requests to web services and business preocesses are queued.

<wlw−manifest>
  <project>
    <async−request−queue>

Syntax

```
<async−request−queue> queueNameString </async−request−queue>
```

Attributes

none.

Hierarchy

Parents: <project>.

Children: none.

<top−level−component>

Defines resources for a top−level component (JWS, JPD or control) of the project.

<wlw−manifest>
  <project>
    <top−level−component>
      <conversation−state−table>
      <external−callbacks>
      <roles−allowed>
      <roles−referenced>
Syntax

```
<top−level−component
    class−name="classString"
    component−type="[ JWS | JPD | Control ]"
    run−as−role="roleString"
>
<!−−
Child elements defining the resources and security configuration for the top−level component.
−−>
</top−level−component>
```

Attributes

Attribute	Description
class−name	Required string. The class name of the EJB
component−type	Required string. Possible values are JWS, JPD, or Control.
run−as−principal	Optional string. A security principal. This class runs with the permissions equivalent to the specified principal.
run−as−role	Optional string. A security role. This class runs with permissions equivalent to the specified role.

Hierarchy

Parents: <project>.

Children: <conversation−state−table>, <external−callbacks>, <roles−allowed>, <roles−referenced>.

<conversation−state−table>

Defines a database table to hold conversation state data for a web service or business process. Adminstrators must create these tables on the server where the component resides.

<wlw−manifest>
  <project>
    <top−level−component>
      <conversation−state−table>

Syntax

```
<conversation−state−table
    table−name="nameString"
>
<!−−
Child elements defining the table's columns.
−−>
</conversation−state−table>
```

Attributes

Attribute	Description
table−name	Required string. Defines the name of the conversational state table.

Hierarchy

Parents: <top−level−component>.

Children: <column>.

<column>

Defines column information for a single column of a conversation state data table.

<wlw−manifest>
  <project>
    <top−level−component>
      <conversation−state−table>
        <column>

Syntax

```
<column>
<!−−
Child elements defining column information.
−−>
</column>
```

Attributes

none.

Hierarchy

Parents: <conversation−state−table>.

Children: <column−name>, <column−type>.

<column−name>

Defines the name of a single column of a conversation state data table.

<wlw−manifest>
  <project>
    <top−level−component>
      <conversation−state−table>
        <column>
          <column−name>

Syntax

```
<column−name> stringColumnName </column−name>
```

Attributes

none.

Hierarchy

Parents: <column>.

Children: none.

<column−type>

Defines the data type of a single column of a conversation state data table.

<wlw−manifest>
  <project>
    <top−level−component>
      <conversation−state−table>
        <column>
          <column−type>

Syntax

```
<column−type> dataTypeString </column−type>
```

Attributes

none.

Hierarchy

Parents: <column>.

Children: none.

<external−callbacks>

Defines information pertaining to call backs directed at this component.

<wlw−manifest>
  <project>
    <top−level−component>
      <external−callbacks>
        <control>
          <role−name>

Syntax

```
<external−callbacks>
<!−−
Child elements defining the control that handles callbacks and the roles required for callback
−−>
</external−callbacks>
```

Attributes

none.

Hierarchy

Parents: <top−level−component>.

Children: <control>.

<control>

Defines the control that handles callbacks for a top−level component.

```
<wlw−manifest>
  <project>
    <top−level−component>
      <external−callbacks>
        <control>
          <role−name>
```

Syntax

```
<control
    control−path="pathString"
>
<!−−
Child element defining the roles neccessary for calling back the component.
−−>
</control>
```

Attributes

Attribute	Description
control−path	Required string. Path to the control.

Hierarchy

Parents: <external−callbacks>.

Children: <role−name>.

<role−name>

Defines a role necessary to callback the top−level component.

```
<wlw−manifest>
  <project>
    <top−level−component>
      <external−callbacks>
        <control>
          <role−name>
```

Syntax

```
<role−name> roleString </role−name>
```

Attributes

none.

Hierarchy

Parents: <control>.

Children: none.

<roles−allowed>

Defines a role necessary to invoke the top−level component. The roles defined here do not need to be identical to the roles required to callback the component.

<wlw−manifest>
  <project>
    <top−level−component>
      <roles−allowed>
        <role−name>

Syntax

```
<roles−allowed>
<!−−
Child elements listing the roles required to invoke the top−level component.
−−>
</roles−allowed>
```

Attributes

none.

Hierarchy

Parents: <top−level−component>.

Children: <role−name>.

<role−name>

A single role required to access a top−level component.

<wlw−manifest>
  <project>
    <top−level−component>

<roles−allowed>
<role−name>

Syntax

```
<role−name> roleString </role−name>
```

Attributes

none.

Hierarchy

Parents: <roles−allowed>.

Children: none.

<roles−referenced>

The top−level component may access other resources using the credentials of the role names specified.

<wlw−manifest>
<project>
<top−level−component>
<roles−referenced>
<role−name>

Syntax

```
<roles−referenced>
<!−−
Child elements listing the roles referenced.
−−>
</roles−referenced>
```

Attributes

none.

Hierarchy

Parents: <top−level−component>.

Children: <role−name>.

<role−name>

Defines one role referenced by the top−level component.

<wlw−manifest>
  <project>
    <top−level−component>
      <roles−referenced>
        <role−name>

Syntax

`<role−name>` *`roleString`* `</role−name>`

Attributes

none.

Hierarchy

Parents: <roles−referenced>.

Children: none.

Example

The following example is the wlw−manifest.xml file generated when selected web services from the project SamplesApp/WebServices are compiled into an EAR file.

```
<?xml version="1.0" encoding="UTF−8"?>
<con:wlw−manifest xmlns:con="http://www.bea.com/2003/03/wlw/config/">
    <con:project name="WebServices">
        <con:async−request−queue>WebServices.queue.AsyncDispatcher</con:async−request−queue>
        <con:async−request−error−queue>WebServices.queue.AsyncDispatcher_error</con:async−reque
        <con:top−level−component class−name="async.Buffer" component−type="JWS">
            <con:conversation−state−table table−name="JWS_WEBSERVICES_ASYNC_BUFFER"/>
            <con:external−callbacks/>
            <con:security−roles>
                <con:role−name xsi:nil="true" xmlns:xsi="http://www.w3.org/2001/XMLSchema−insta
            </con:security−roles>
        </con:top−level−component>
        <con:top−level−component class−name="async.Conversation" component−type="JWS">
            <con:conversation−state−table table−name="JWS_ERVICES_ASYNC_CONVERSATION"/>
            <con:external−callbacks>
                <con:control control−path="helloAsync$callback"/>
            </con:external−callbacks>
            <con:security−roles>
                <con:role−name xsi:nil="true" xmlns:xsi="http://www.w3.org/2001/XMLSchema−insta
            </con:security−roles>
        </con:top−level−component>
        <con:top−level−component run−as−principal="weblogic" run−as−role="Administrators" class
            <con:external−callbacks/>
            <con:security−roles>
                <con:role−name>Administrators</con:role−name>
            </con:security−roles>
        </con:top−level−component>
        <con:top−level−component class−name="security.roleBased.VeriCheck" component−type="JWS"
            <con:conversation−state−table table−name="JWS_CURITY_ROLEBASED_VERICHECK"/>
            <con:external−callbacks>
```

```
                <con:control control-path="bankControl$callback"/>
            </con:external-callbacks>
            <con:security-roles>
                <con:role-name xsi:nil="true" xmlns:xsi="http://www.w3.org/2001/XMLSchema-insta
            </con:security-roles>
        </con:top-level-component>
    </con:project>
</con:wlw-manifest>
```

Related Topics

wlwBuild Command

wlw-config.xml Configuration File

WS−Security Policy File Reference (WSSE File Reference)

The WS−Security policy file (WSSE file) defines the security policy applied to the SOAP messages that pass between web services and their clients. Three security mechanisms can be defined in a WSSE file: (1) security tokens, (2) digital signatures, and (3) encryption.

Incoming secured SOAP messages are first processed by WebLogic Workshop in accordance with the WSSE file. If the incoming SOAP message passes the security regimen defined in the WSSE file, then the message is passed to the target web service or control for normal processing.

Outgoing SOAP messages are first secured by WebLogic Workshop according to the policy defined in the WSSE file before they are sent out over the wire. That is, outgoing SOAP messages are enhanced with any security tokens, signatures, and encryption specified in the WSSE file before they are sent.

General Structure

```
<wsSecurityPolicy>
  <wsSecurityIn>
    <token>
    <encryptionRequired>
      <decryptionKey>
        <alias>
        <password>
    <signatureRequired>
  <wsSecurityOut>
    <userNameToken>
      <userName>
      <password>
    <encryption>
      <encryptionKey>
        <alias>
        <x509Certificate>
      <useInboundSignatureCertificate>
    <signatureKey>
      <alias>
      <password>
    <additionalSignedElements>
      <secureElement>
    <additionalEncryptedElements>
      <secureElement>
  <keyStore>
    <keyStoreLocation>
    <keyStorePassword>
```

<wsSecurityPolicy>

The top level element within a Web Service Security file (WSSE file). Describes the web service security

policy for a request/response pair.

Syntax

```
<wsSecurityPolicy
    xmlns="nameSpace"
>
```

Attribute	Description
xmlns	Specifies the name space for the WSSE policy file.

Hierarchy

Parents: none.

Children: <wsSecurityIn>, <wsSecurityOut>, <keyStore>.

<wsSecurityIn>

Defines the security regimen to be applied to incoming SOAP messages to the web service. SOAP message can be checked for three sorts of security enhancements: (1) security tokens (see the <token> element), (2) digital signature (see the <signatureRequired> element), and (3) encryption (see the <encryptionRequired> element).
<wsSecurityPolicy>
  <wsSecurityIn>

Syntax

```
<wsSecurityIn>
    <!--
    Elements that describe the security regimen to be applied to
    incoming SOAP messages.
    -->
</wsSecurityIn>
```

Hierarchy

Parents: <wsSecurityPolicy>.

Children: <token>, <signaturedRequired>, <encryptionRequired>.

<token>

When this element is present, then a valid token is expected in the inbound message. This token must successfully match up with a principal in the WLS Security Framework.

<wsSecurityPolicy>
  <wsSecurityIn>

<token>

Syntax

```
<token
    tokenType="username"
>
```

Attribute	Description
tokenType	Required string. It has one valid value: "username". When this element is present, a username/password token must be present in inbound SOAP messages. The username/password must successfully match a principal known to WebLogic Server.

Hierarchy

Parents: <wsSecurityIn>.

Children: none.

<encryptionRequired>

When this element is present, inbound SOAP messages must be encrypted. This element must be accompanied by a <decryptionKey> child element**.**

Syntax

```
<encryptionRequired>
    <!--
    Contains information for decrypting the incoming encrypted SOAP message.
    -->
</encryptionRequired>
```

Hierarchy

Parents: <wsSecurityIn>.

Children: <decryptionKey>.

<decryptionKey>

This element contains information to resolve the decryption key pair. The private key is used to decrypt the incoming message. The public key (contained within the X509 Digital Certificate) is put into the WSDL so that clients of the web service can use it to encrypt SOAP messages.

The child elements, <alias> and <password> are provided for access to the private key.

```
<wsSecurityPolicy>
  <wsSecurityIn>
    <encryptionRequired>
      <decryptionKey>
```

<alias>
<password>

Syntax

```
<decryptionKey>
    <!--
    Information for retrieving the private key from the keystore.
    -->
</decryptionKey>
```

Hierarchy

Parents: <encryptionRequired>.

Children: <alias>, <password>.

<alias>

Specifies the alias used to look up the private key (or key pair) in the keystore.

<wsSecurityPolicy>
<wsSecurityIn>
<encryptionRequired>
<decryptionKey>
<alias>

Syntax

```
<alias> stringAlias </alias>
```

Hierarchy

Parents: <decryptionKey>

Children: none.

<password>

This element specifies the password used to look up the private key in the keystore. (Note this element does not have the same syntax as the <password> element used as the child of the <userNameToken> element.)

Note: Passwords appear in plain text in the WSSE policy file. For tools to encrypt these passwords, see Securing WS−Security Passwords.

<wsSecurityPolicy>
<wsSecurityIn>
<encryptionRequired>
<decryptionKey>
<password>

Syntax

```
<password> stringPassword </password>
```

Hierarchy

Parents: <decryptionKey>.

Children: none.

<signatureRequired>

Takes boolean values. If this the value of this element is true, then inbound SOAP messages are expected to be signed.

<wsSecurityPolicy>
  <wsSecurityIn>
    <signatureRequired>

Syntax

```
<signaturedRequired> boolean </signaturedRequired>
```

Hierarchy

Parents: <wsSecurityIn>

Children: none.

<wsSecurityOut>

This element defines how outgoing SOAP messages should be secured before they are sent out over the wire. Three security enhancements can be applied to outgoing messages: (1) security tokens (see the <userNameToken> element), (2) digital signature (see the <signatureKey> element), and (3) encryption (see the <encryption> element).

<wsSecurityPolicy>
  <wsSecurityOut>
    <userNameToken>
    <encryption>
    <signatureKey>
    <additionalSignedElements>
    <additionalEncryptedElements>

Syntax

```
<wsSecurityOut>
         <!--
         Children elements that describe how to enhance
         outbound SOAP messages.
```

```
        -->
    </wsSecurityOut>
```

Hierarchy

Parents: <wsSecurityPolicy>

Children: <userNameToken>, <encryption>, <signatureKey>, <additionalSignedElements>, <additionalEncryptedElements>

<userNameToken>

When this element is present, outbound SOAP messages are enhanced with a username and password token. This element must have two children elements: <userName> and <password>.

<wsSecurityPolicy>
  <wsSecurityOut>
    <userNameToken>

Syntax

```
<userNameToken>
     <!--
     Children elements specifying the username and password
     to include with the outbound SOAP message
     -->
</userNameToken>
```

Hierarchy

Parents: <wsSecurityOut>.

Children: <userName>, <password>.

<userName>

Takes String values. The String must correspond to a username in the security realm associated with the web service's running WebLogic Server domain.

<wsSecurityPolicy>
  <wsSecurityOut>
    <userNameToken>
      <userName>

Syntax

```
<userName> stringUserName </userName>
```

Hierarchy

Parents: <userNameToken>

Children: none.

<password>

This password must be the correct password of the user in the security realm associated with the web service's running WebLogic Domain.

Note: Passwords appear in plain text in the WSSE policy file. For tools to encrypt these passwords, see Securing WS−Security Passwords.

```
<wsSecurityPolicy>
  <wsSecurityOut>
    <userNameToken>
      <password>
```

Attribute	Description
type	Required string. There is one valid value: "TEXT".

Syntax

```
<password> stringPassword </password>
```

Hierarchy

Parents: <userNameToken>.

Children: none.

<encryption>

If this element is present then the body of the SOAP message will be encrypted on the way out. The SOAP recipient's public key used for encryption can be designated by directly providing the public key or using the public key that was provided for the inbound Signature (assuming that there is a previous inbound SOAP message containing a digital signature). The <encryption> element must have the child element <encryptionKey> or <useInboundSignatureCertificate>.

To sign elements in the SOAP header, see <additionalEncryptedElements>.

```
<wsSecurityPolicy>
  <wsSecurityOut>
    <encryption>
      <encryptionKey>
      <useInboundSignatureCertificate>
```

Syntax

```
<encryption>
     <!--
     Child elements specifying how to encrypt the outbound SOAP message.
     -->
</encryption>
```

Hierarchy

Parents: <wsSecurityOut>

Children: <encryptionKey>, <useInboundSignatureCertificate>

<encryptionKey>

This is the key used to encrypt the outbound SOAP message. In the case where the web service control is initiating a conversation with another web service the x509Certificate may have been initially created from the target web service's WSDL file.

<wsSecurityPolicy>
  <wsSecurityOut>
    <encryption>
      <encryptionKey>

Syntax

```
<encryptionKey>
<!--
Child element specifying either that the public key should be retrieved from the keystore,
or from a x509 cert.
-->
</encryptionKey>
```

Hierarchy

Parents: <encryption>

Children: <alias>, <x509Certificate>.

<alias>

Specifies the alias used to look up the public key in the keystore when retrieving a key for encryption of outbound SOAP messages.

<wsSecurityPolicy>
  <wsSecurityOut>
    <encryption>
      <encryptionKey>
        <alias>

Syntax

```
<alias> stringAilas </alias>
```

Hierarchy

Parents: <encryptionKey>.

Children: none.

<x509Certificate>

Specifies the base64 encoded X509 certificate for the purpose of the retrieving the public key from it.

```
<wsSecurityPolicy>
  <wsSecurityOut>
    <encryption>
      <encryptionKey>
        <x509Certificate>
```

Syntax

```
<x509Certificate> base64EncodedCert </x509Certificate>
```

Hierarchy

Parents: <encryptionKey>.

Children: none.

<useInboundSignatureCertificate>

Specifies that outbound SOAP message should be encrypted with the public key in the inbound SOAP message's x509 certificate. If this element is omitted the value is false.

```
<wsSecurityPolicy>
  <wsSecurityOut>
    <encryption>
      <useInboundSignatureCertificate>
```

Syntax

```
<useInboundSignatureCertificate> boolean </useInboundSignatureCertificate>
```

Hierarchy

Parents: <encryption>.

Children: none.

<signatureKey>

This element is used to designate the private key that will be used for signing. This keypair should also have a certificate associated with so that the validation certificate can be added to the message. If this element is present, only the SOAP body is signed. To sign header elements, see <addtionalSignedElements>.

<wsSecurityPolicy>
  <wsSecurityOut>
    <signatureKey>
      <alias>
      <password>

Syntax

```
<signatureKey>
     <!--
     Child elements specifying the alias and password used to retrieve the signing private key
     -->
</signatureKey>
```

Hierarchy

Parents: <wsSecurityOut>.

Children: <alias>, <password>.

<alias>

This alias is used to look up the private key (or key pair) in the keystore.

<wsSecurityPolicy>
  <wsSecurityOut>
    <signatureKey>
      <alias>

Syntax

```
<alias> stringAlias </alias>
```

Hierarchy

Parents: <signatureKey>.

Children: none.

<password>

The password associated with the alias used to retrieve the private key from the keystore.

Note: Passwords appear in plain text in the WSSE policy file. For tools to encrypt these passwords, see Securing WS−Security Passwords.

<wsSecurityPolicy>
  <wsSecurityOut>
    <signatureKey>
      <password>

Syntax

```
<password> stringPassword </password>
```

Hierarchy

Parents: <signatureKey>.

Children: none.

<additionalSignedElements>

Contains namespace and element pairs that need to be secured. Note that only the SOAP body is signed, if a signature is called for by a <signatureKey> element. <additionalSignedElements> is used to specify SOAP header elements for signing.

<wsSecurityPolicy>
  <wsSecurityOut>
    <additionalSignedElements>
      <secureElement>

Syntax

```
<additionalSignedElements>
     <!--
     Child elements listing the elements to be signed.
     -->
</additionalSignedElements>
```

Hierarchy

Parents: <wsSecurityOut>.

Children: <secureElement>.

<secureElement>

Specifies an element to be signed.

Attribute	Description
name	Required string. Names an XML element to be signed.

nameSpace	Required string. Any URI value.

<wsSecurityPolicy>
  <wsSecurityOut>
    <additionalSignedElements>
      <secureElement>

Syntax

```
<secureElement
    name="stringName"
    nameSpace="stringURI"
>
```

Hierarchy

Parents: <additionalSignedElements>.

Children: none.

<additionalEncryptedElements>

Specifies an element to be encrypted. Note that the <encryption> element encrypts only the SOAP body by default. Use <additionalEncryptedElements> to encrypt sensitive elements in the SOAP header.

<wsSecurityPolicy>
  <wsSecurityOut>
    <additionalEncryptedElements>
      <secureElement>

Syntax

```
<additionalEncryptedElements>
     <!--
     Child elements listing the elements to be encrypted.
     -->
</additionalEncryptedElements>
```

Hierarchy

Parents: <wsSecurityOut>.

Children: <secureElement>.

<secureElement>

Specifies an element to be encrypted.

Attribute	Description
name	Required string. Names an XML element to be encrypted.

nameSpace	Required string. Any URI value.

<wsSecurityPolicy>
  <wsSecurityOut>
    <additionalEncryptedElements>
      <secureElement>

Syntax

```
<secureElement
    name="stringName"
    nameSpace="stringURI"
>
```

Hierarchy

Parents: <additionalEncryptedElements>.

Children: none.

<keyStore>

This element is optional. If a keystore is not designated then Weblogic Server's default keystore is used. If present, the <keyStore> element must have the child elements <keyStoreLocation> and <keyStorePassword>.

<wsSecurityPolicy>
  <keyStore>
    <keyStoreLocation>
    <keyStorePassword>

Syntax

```
<keyStore>
     <!--
     Child elements specifying the keystore location and password.
     -->
</keyStore>
```

Hierarchy

Parents: <wsSecurityPolicy>.

Children: <keyStoreLocation>, <keyStorePassword>.

<keyStoreLocation>

The <keyStoreLocation> element specifies the path to the keystore. An absolute path or relative path can be designated. If a relative path is used then the base directory is the active Weblogic Server domain directory (e.g., BEA_HOME\weblogic81\samples\domains\workshop is the samples domain directory).

<wsSecurityPolicy>
  <keyStore>
    <keyStoreLocation>

Syntax

```
<keyStoreLocation> stringPath </keyStoreLocation>
```

Hierarchy

Parents: <keyStore>.

Children: none.

<keyStorePassword>

The <keyStorePassword> specifies the password required to access the keystore.

Note: Passwords appear in plain text in the WSSE policy file. For tools to encrypt these passwords, see Securing WS−Security Passwords.

<wsSecurityPolicy>
  <keyStore>
    <keyStorePassword>

Syntax

```
<keyStorePassword> stringPassword </keyStorePassword>
```

Hierarchy

Parents: <keyStore>.

Children: none.

Examples

Example #1

The following WSSE policy file requires that inbound SOAP messages must be encrypted.

```
<wsSecurityPolicy xsi:schemaLocation="WSSecurity−policy.xsd" xmlns="http://www.bea.com/2003/03/
    <wsSecurityIn>
        <encryptionRequired>
            <decryptionKey>
                <alias>companyB</alias>
                <password>{3DES}OZyVt5STUU8BMMJLPGSkYQ==</password>
            </decryptionKey>
        </encryptionRequired>
    </wsSecurityIn>
    <keyStore>
```

```
        <keyStoreLocation>wlwwsse.jks</keyStoreLocation>
        <keyStorePassword>{3DES}OZyVt5STUU8BMMJLPGSkYQ==</keyStorePassword>
    </keyStore>
</wsSecurityPolicy>
```

Example #2

The following WSSE policy file requires that inbound SOAP messages include a username token and be encrypted. Outbound messages are encrypted before they are sent out over the wire.

```
<wsSecurityPolicy xsi:schemaLocation="WSSecurity-policy.xsd" xmlns="http://www.bea.com/2003/03/
        <wsSecurityIn>
                <token tokenType="username"/>
                <encryptionRequired>
                        <decryptionKey>
                                <alias>companyB</alias>
                                <password>{3DES}OZyVt5STUU8BMMJLPGSkYQ==</password>
                        </decryptionKey>
                </encryptionRequired>
                <signatureRequired>true</signatureRequired>
        </wsSecurityIn>
        <wsSecurityOut>
                <encryption>
                        <encryptionKey>
                                <alias>companyB</alias>
                        </encryptionKey>
                </encryption>
        </wsSecurityOut>
        <keyStore>
                <keyStoreLocation>wlwwsse.jks</keyStoreLocation>
                <keyStorePassword>{3DES}OZyVt5STUU8BMMJLPGSkYQ==</keyStorePassword>
        </keyStore>
</wsSecurityPolicy>
```

Example #3

The following WSSE policy file requires that inbound SOAP messages include a (1) username/password token, (2) a signature, and (3) be encrypted.

```
<wsSecurityPolicy xsi:schemaLocation="WSSecurity-policy.xsd" xmlns="http://www.bea.com/2003/03/
        <wsSecurityIn>
                <token tokenType="username"/>
                <encryptionRequired>
                        <decryptionKey>
                                <alias>companyB</alias>
                                <password>{3DES}OZyVt5STUU8BMMJLPGSkYQ==</password>
                        </decryptionKey>
                </encryptionRequired>
                <signatureRequired>true</signatureRequired>
        </wsSecurityIn>
        <keyStore>
                <keyStoreLocation>wlwwsse.jks</keyStoreLocation>
                <keyStorePassword>{3DES}OZyVt5STUU8BMMJLPGSkYQ==</keyStorePassword>
        </keyStore>
</wsSecurityPolicy>
```

Example #4

The <encryption> element directs that the body of outgoing SOAP messages be encrypted and, additionally, the <additionalEncrptedElements> element directs that the <person> element in the SOAP header be encrypted.

```
<wsSecurityPolicy xsi:schemaLocation="WSSecurity−policy.xsd" xmlns="http://www.bea.com/2003/03/
    <wsSecurityOut>
        <encryption>
            <encryptionKey>
                <alias>companyB</alias>
            </encryptionKey>
        </encryption>
        <additionalEncryptedElements>
            <secureElement name="person" namespace="http://www.bea.com/person"/>
        </additionalEncryptedElements>
    </wsSecurityOut>
    <keyStore>
        <keyStoreLocation>wlwwsse.jks</keyStoreLocation>
        <keyStorePassword>{3DES}OZyVt5STUU8BMMJLPGSkYQ==</keyStorePassword>
    </keyStore>
</wsSecurityPolicy>
```

Related Topics

Applying WS−Security Policy Files

WS−Security File Elements