BEA WebLogic® Event Server

WebLogic Event Server Reference

Version 2.0
July 2007

Contents

3. Deployer Command-Line Reference

4. Metadata Annotations

5. XSD Schema Reference for WebLogic Event Server Files

CHAPTER 1

Introduction and Roadmap

This section describes the contents and organization of this guide—*WebLogic Event Server Reference*.

Document Scope and Audience

This document is a resource for software developers who develop event driven real-time applications. It also contains information that is useful for business analysts and system architects who are evaluating WebLogic Event Server or considering the use of WebLogic Event Server for a particular application.

The topics in this document are relevant during the design, development, configuration, deployment, and performance tuning phases of event driven applications. The document also includes topics that are useful in solving application problems that are discovered during test and pre-production phases of a project.

It is assumed that the reader is familiar with the Java programming language and Spring.

WebLogic Event Server Documentation Set

This document is part of a larger WebLogic Event Server documentation set that covers a comprehensive list of topics. The full documentation set includes the following documents:

- *Getting Started With WebLogic Event Server*
- *Creating WebLogic Event Server Applications*
- *WebLogic Event Server Administration and Configuration Guide*
- *EPL Reference Guide*
- *WebLogic Event Server Reference Guide*
- *WebLogic Event Server Release Notes*

See the main WebLogic Event Server documentation page for further details.

Guide to This Document

This document is organized as follows:

- This chapter, Chapter 1, "Introduction and Roadmap," introduces the organization of this guide and the WebLogic Event Server documentation set and samples.
- Chapter 2, "WebLogic Event Server Spring Tag Reference," lists the custom Spring tags you can use in a variety of XML configuration files to further configure WebLogic Event Server applications.
- Chapter 3, "Deployer Command-Line Reference," provides reference information for the Deployer tool used to install, update, start and stop OSGi bundles to WebLogic Event Server.
- Chapter 4, "Metadata Annotations," provides reference information for the metadata annotations you can use in your adapter Java file to access its configuration.
- Chapter 5, "XSD Schema Reference for WebLogic Event Server Files," provides the XSD Schemas for the various WebLogic Event Server XML files, including component configuration files, EPN assembly file, deployments file, and server configuration file.

Samples for the WebLogic Event Server Application Developer

In addition to this document, BEA Systems provides a variety of code samples for WebLogic Event Server application developers. The examples illustrate WebLogic Event Server in action, and provide practical instructions on how to perform key development tasks.

BEA recommends that you run some or all of the examples before programming and configuring your own event driven application.

The examples are distributed in two ways:

- Pre-packaged and compiled in their own domain so you can immediately run them after you install the product.
- Separately in a Java source directory so you can see a typical development environment setup.

The following two examples are provided in both their own domain and as Java source in this release of WebLogic Event Server:

- HelloWorld—Example that shows the basic elements of a WebLogic Event Server application. See Hello World Example for additional information.

  The HelloWorld domain is located in *WLEVS_HOME*\samples\domains\helloworld_domain, where *WLEVS_HOME* refers to the top-level WebLogic Event Server directory, such as c:\beahome\wlevs20.

  The HelloWorld Java source code is located in *WLEVS_HOME*\samples\source\applications\helloworld.

- ForeignExchange (FX)—Example that includes multiple adapters, streams, and complex event processor with a variety of EPL rules, all packaged in the same WebLogic Event Server application. See Foreign Exchange (FX) Example for additional information.

  The ForeignExchange domain is located in *WLEVS_HOME*\samples\domains\fx_domain, where *WLEVS_HOME* refers to the top-level WebLogic Event Server directory, such as c:\beahome\wlevs20.

  The ForeignExchange Java source code is located in *WLEVS_HOME*\samples\source\applications\fx.

WebLogic Event Server also includes an algorithmic trading application, pre-assembled and deployed in its own sample domain; the source code for the example, however, is not provided.

The algorithmic trading domain is located in *WLEVS_HOME*\samples\domains\algotrading_domain.

CHAPTER 2

WebLogic Event Server Spring Tag Reference

This section contains information on the following subjects:

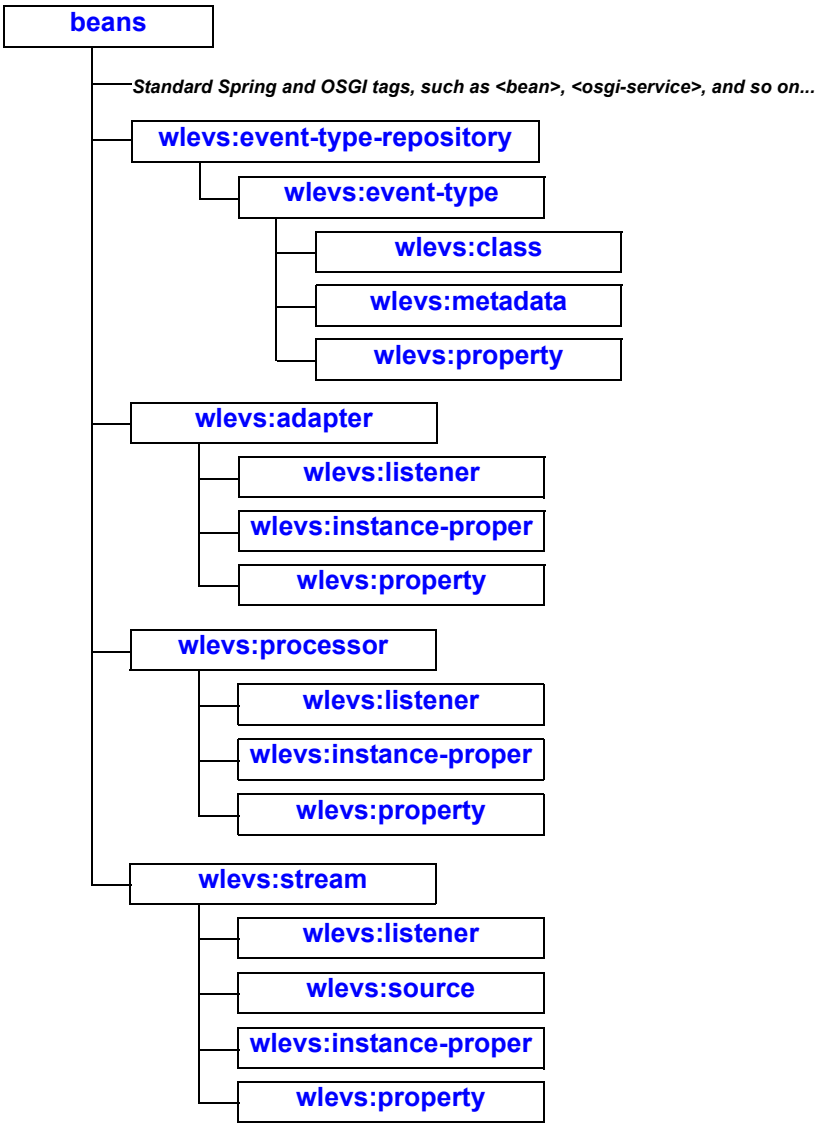
Overview of the WebLogic Event Server Spring Tags

WebLogic Event Server provides a number of Spring tags that you use in the EPN assembly file of your application to register event types, declare the components of the event processing network and specify how they are linked together. The EPN assembly file is an extension of the standard Spring context file.

Graphical Representation

The following graphic describes the hierarchy of the WebLogic Event Server Spring tags.

Figure 2-1 Hierarchy of WebLogic Event Server Spring Tags



Example of an EPN Assembly File That Uses WebLogic Event Server Tags

The following sample EPN assembly file from the HelloWorld application shows how to use many of the WebLogic Event Server tags:

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xmlns:osgi="http://www.springframework.org/schema/osgi"
       xmlns:wlevs="http://www.bea.com/ns/wlevs/spring"
       xsi:schemaLocation="
  http://www.springframework.org/schema/beans
  http://www.springframework.org/schema/beans/spring-beans.xsd
  http://www.springframework.org/schema/osgi
  http://www.springframework.org/schema/osgi/spring-osgi.xsd
  http://www.bea.com/ns/wlevs/spring
  http://www.bea.com/ns/wlevs/spring/spring-wlevs.xsd">

    <!-- First, create and register the adapter (and factory) that generates
hello world messages -->
    <osgi:service interface="com.bea.wlevs.ede.api.AdapterFactory">
        <osgi:service-properties>
            <prop key="type">hellomsgs</prop>
        </osgi:service-properties>
        <bean
class="com.bea.wlevs.adapter.example.helloworld.HelloWorldAdapterFactory" />
    </osgi:service>

    <wlevs:event-type-repository>
        <wlevs:event-type type-name="HelloWorldEvent">

<wlevs:class>com.bea.wlevs.event.example.helloworld.HelloWorldEvent</wlevs:cla
ss>
        </wlevs:event-type>
    </wlevs:event-type-repository>

    <!-- Assemble EPN (event processing network) -->

    <!-- The adapter id is used by the configuration system, so needs to be
well-known -->
    <wlevs:adapter id="helloworldAdapter" provider="hellomsgs"
manageable="true">
        <!-- This property is also configure by dynamic config -->
        <wlevs:instance-property name="message" value="HelloWorld - the
currenttime is:"/>
    </wlevs:adapter>
```

```
    <!-- The processor id is used by the configuration system, so needs to be
well-known -->
    <wlevs:processor id="helloworldProcessor" manageable="true" />

    <wlevs:stream id="helloworldInstream" manageable="true">
        <wlevs:listener ref="helloworldProcessor"/>
        <wlevs:source ref="helloworldAdapter"/>
    </wlevs:stream>

    <!-- Manageable is so that we can monitor the event throughput -->
    <wlevs:stream id="helloworldOutstream" manageable="true">
        <wlevs:listener>
            <!-- Create business object -->
            <bean class="com.bea.wlevs.example.helloworld.HelloWorldBean"/>
        </wlevs:listener>
        <wlevs:source ref="helloworldProcessor"/>
    </wlevs:stream>

</beans>
```

wlevs:adapter

Use this tag to declare an adapter service to the Spring application context.

Child Tags

The `wlevs:adapter` Spring tag supports the following child tags:

- wlevs:listener
- wlevs:instance-property
- wlevs:property

Attributes

The following table lists the attributes of the `wlevs:adapter` Spring tag.

Table 2-1 Attributes of the wlevs:adapter Spring Tag

Attribute	Description	Data Type	Required?
id	Unique identifier for this component.	String	Yes.
advertise	Advertises this service in the OSGi registry. Valid values are `true` and `false`. Default value is `false`.	Boolean	No.
depends-on	The name of the Spring beans that the underlying Spring bean that implements this componet depends on. Use this attribute to enforce the correct initialization order of beans in your application. If you do not require any specific initialization order, then do not specify this attribute. If using, set this attribute to the value of the `id` attribute of the dependent Spring bean.	String	No.
lazy-init	Specifies whether WeblogicEvent Server should lazily initialize the underlying Spring bean that implements this component. If set to `false`, the bean will be instantiated on startup by bean factories that perform eager initialization of singletons. Valid values for this attribute are `true` and `false`. The default value is `false`.	Boolean.	No.
listeners	Specifies the components that listen to this component. Set this attribute to the value of the `id` attribute of the element that declared the component.	String	No.

Table 2-1 Attributes of the wlevs:adapter Spring Tag

Attribute	Description	Data Type	Required?
provider	Specifies the adapter service provider. Typically the value of this attribute is a reference to the OSGi-registered adapter factory service. If you are using the `csvgen` or `loadgen` utilities to simulate a data feed, use the hard-coded `csvgen` or `loadgen` values, respectively, such as: `provider="csvgen"`	String	Yes
manageable	Specifies that this component can be monitored using the WebLogic Event Server monitoring framework. Setting this attribute to `true` may adversely impact the performance of your application, so change the default setting of this attribute only if you are sure you want monitoring information about this component. Valid values for this attribute are `true` and `false`. The default value is `false`.	Boolean	No.

Example

The following example shows how to use the `wlevs:adapter` tag in the EPN assembly file:

```
<wlevs:adapter id="helloworldAdapter" provider="hellomsgs">
    <wlevs:instance-property name="message"
                             value="HelloWorld - the current time is:"/>
</wlevs:adapter>
```

In the example, the adapter's unique identifier is `helloworldAdapter`. The provider is an OSGi service, also registered in the EPN assembly file, whose reference is `hellomsgs`. The adapter has a static property called `message`, which implies that the adapter Java file has a `setMessage()` method.

wlevs:class

Specifies the fully qualified JavaBean classname that implements a particular event type.

This tag is used only as a child of wlevs:event-type.

This tag has no child tags and no attributes

Example

The following example shows how to use the `wlevs:class` tag in the EPN assembly file:

```
 <wlevs:event-type type-name="HelloWorldEvent">

<wlevs:class>com.bea.wlevs.event.example.helloworld.HelloWorldEvent</wlevs
:class>
 </wlevs:event-type>
```

In the example, the `<wlevs:class>` tag specifies the class (`com.bea.wlevs.event.example.helloworld.HelloWorldEvent`) that defines the `HelloWorldEvent` event type.

wlevs:event-type-repository

Use this tag to group together one or more `wlevs:event-type` tags, each of which is used to register an event type used throughout the application.

This tag does not have any attributes.

Child Tags

The `wlevs:event-type-repository` Spring tag supports the following child tag:

- wlevs:event-type

Example

The following example shows how to use the `wlevs:event-type-repository` tag in the EPN assembly file:

```
<wlevs:event-type-repository>
 <wlevs:event-type type-name="HelloWorldEvent">

<wlevs:class>com.bea.wlevs.event.example.helloworld.HelloWorldEvent</wlevs
:class>
 </wlevs:event-type>
</wlevs:event-type-repository>
```

In the example, the `<wlevs:event-type-repository>` tag groups a single `<wlevs:event-type>` tag to declare a single event type: `HelloWorldEvent`. See "wlevs:event-type" on page 2-8 for additional details.

wlevs:event-type

Specifies the definition of an event type used in the WebLogic Event Server application. Once you define the event types of the application, you can reference them in the adapter and business class POJO, as well as the EPL rules.

You can define an event type in the following ways:

- Create a JavaBean class that represents your event type and specify its fully qualified classname using the wlevs:class child tag.
- Use the wlevs:metadata child tag to list the properties of the data type and allow WebLogic Event Server to automatically create the Java class at runtime.

You can specify one of *either* `wlevs:class` or `wlevs:metadata` as a child of `wlevs:event-type`, but not both.

You can also use the wlevs:property child tag to specify a custom property to apply to the event type.

BEA recommends that you define your event type by using the `wlevs:class` child tag because you can them reuse the specified JavaBean class, and you control exactly what the event type looks like.

Child Tags

The `wlevs:event-type` Spring tag supports the following child tags:

- wlevs:class
- wlevs:metadata
- wlevs:property

Attributes

The following table lists the attributes of the `wlevs:event-type` Spring tag.

Table 2-2 Attributes of the wlevs:link Spring Tag

Attribute	Description	Data Type	Required?
id	Specifies the unique identifier for this event type. If you do not specify this attribute, WebLogic Event Server automatically generates an identifier for you.	q	No.
type-name	Specifies the name of of this event type. This is the name you use whenever you reference the event type in the adapter, business POJO, or EPL rules.	String	Yes.

Example

The following example shows how to use the `wlevs:event-type` tag in the EPN assembly file:

```
<wlevs:event-type-repository>
 <wlevs:event-type type-name="HelloWorldEvent">

<wlevs:class>com.bea.wlevs.event.example.helloworld.HelloWorldEvent</wlevs
:class>
 </wlevs:event-type>
</wlevs:event-type-repository>
```

In the example, the name of the event type is `HelloWorldEvent` and its definition is determined by the `com.bea.wlevs.event.example.helloworld.HelloWorldEvent` JavaBean class.

wlevs:instance-property

Specifies the properties that apply to the create stage instance of the component to which this is a child tag. This allows declarative configuration of user-defined stage properties.

This tag is used only as a child of wlevs:adapter, wlevs:processor, or wlevs:stream.

The `wlevs:instance-property` tag is defined as the Spring `propertyType` type; for additional details of this Spring data type, the definition of the allowed child tags, and so on, see the Spring 2.0 XSD.

Child Tags

You can specify one of the following standard Spring tags as a child tag of the `wlevs:instance-property` tag:

- `meta`
- `bean`
- `ref`
- `idref`
- `value`
- `null`
- `list`
- `set`
- `map`
- `props`

Attributes

The following table lists the attributes of the `wlevs:instance-property` Spring tag.

Table 2-3 Attributes of the wlevs:instance-property Spring Tag

Attribute	Description	Data Type	Required?
name	Specifies the name of the property, following JavaBean naming conventions.	String	Yes.
ref	A short-cut alternative to a nested `<ref bean='...'/>` element.	String	No.
value	A short-cut alternative to a nested `<value>...</value>` element.	String	No.

Example

The following example shows how to use the `wlevs:instance-property` tag in the EPN assembly file:

```
<wlevs:adapter id="helloworldAdapter" provider="hellomsgs">
   <wlevs:instance-property name="message" value="HelloWorld - the current
time is:"/>
</wlevs:adapter>
```

In the example, the bean that implements the `helloworldAdapter` adapter component expects an instance property called `message`; the sample `wlevs:instance-property` tag above sets the value of this property to `HelloWorld - the current time is:`.

wlevs:listener

Specifies the component that listens to the component to which this tag is a child. A listener can be an instance of any other component. You can also nest the definition of a component within a particular `wlevs:listener` component to specify the component that listens to the parent.

WARNING: Nested definitions are not eligible for dynamic configuration or monitoring.

This tag is always a child of wlevs:adapter, wlevs:processor, or wlevs:stream.

Attributes

The following table lists the attributes of the `wlevs:listener` Spring tag.

Table 2-4 Attributes of the wlevs:listener Spring Tag

Attribute	Description	Data Type	Required?
ref	Specifies the component that listens to the parent component . Set this attribute to the value of the `id` attribute of the listener component. You do not specify this attribute if you are nesting listeners.	String	No.

Example

The following example shows how to use the `wlevs:listener` tag in the EPN assembly file:

```
<wlevs:processor id="helloworldProcessor">
    <wlevs:listener ref="helloworldOutstream"/>
</wlevs:processor>
```

In the example, the `hellworldOutstream` component listens to the `helloworldProcessor` component. It is assumed that the EPN assembly file also contains a declaration for a `<wlevs:adapter>`, `<wlevs:stream>`, or `<wlevs:processor>` component whose unique identifier is `helloworldOustream`.

wlevs:metadata

Specifies the definition of an event type by listing its fields as a group of Spring `entry` tags. When you define an event type this way, WebLogic Event Server automatically generates the Java class for you.

Use the `key` attribute of the `entry` tag to specify the name of a field and the `value` attribute to specify the Java class that represents the field's data type.

This tag is used only as a child of wlevs:event-type.

The `wlevs:metadata` tag is defined as the Spring `mapType` type; for additional details of this Spring data type, see the Spring 2.0 XSD.

Child Tags

The `wlevs:metadata` tag can have one or more standard Spring `entry` child tags.

Attributes

The following table lists the attributes of the `wlevs:metadata` Spring tag.

Table 2-5 Attributes of the wlevs:metadata Spring Tag

Attribute	Description	Data Type	Required?
key-type	The default fully qualified classname of a Java data type for nested `entry` tags. You use this attribute *only* if you have nested `entry` tags.	String	No.

Example

The following example shows how to use the `wlevs:metadata` tag in the EPN assembly file:

```
<wlevs:event-type type-name="ForeignExchangeEvent">
    <wlevs:metadata>
        <entry key="symbol" value="java.lang.String"/>
        <entry key="price" value="java.lang.Double"/>
        <entry key="fromRate" value="java.lang.String"/>
        <entry key="toRate" value="java.lang.String"/>
    </wlevs:metadata>
    ...
</wlevs:event-type>
```

In the example, the `wlevs:metadata` tag groups together four standard Spring `entry` tags that represent the four fields of the `ForeignExchangeEvent`: `symbol`, `price`, `fromRate`, and `toRate`. The data types of the fields are `java.lang.String`, `java.lang.Double`, `java.lang.String`, and `java.lang.String`, respectively.

wlevs:processor

Use this tag to declare a processor to the Spring application context.

Child Tags

The `wlevs:processor` Spring tag supports the following child tags:

- wlevs:instance-property
- wlevs:listener
- wlevs:property

Attributes

The following table lists the attributes of the `wlevs:processor` Spring tag.

Table 2-6 Attributes of the wlevs:processor Spring Tag

Attribute	Description	Data Type	Required?
id	Unique identifier for this component. This identifier also corresponds to the `<name>` element in the XML configuration file for this processor; this is how WebLogic Event Server knows which EPL rules to execute for which processor component in your network.	String	Yes.
advertise	Advertises this service in the OSGi registry. Valid values are `true` and `false`. Default value is `false`.	Boolean	No.
depends-on	The name of the Spring beans that the underlying Spring bean that implements this componet depends on. Use this attribute to enforce the correct initialization order of beans in your application. If you do not require any specific initialization order, then do not specify this attribute. If using, set this attribute to the value of the `id` attribute of the dependent Spring bean.	String	No.
lazy-init	Specifies whether WeblogicEvent Server should lazily initialize the underlying Spring bean that implements this component. If set to `false`, the bean will be instantiated on startup by bean factories that perform eager initialization of singletons. Valid values for this attribute are `true` and `false`. The default value is `false`.	Boolean.	No.
listeners	Specifies the components that listen to this component. Set this attribute to the value of the `id` attribute of the element that declared the component.	String	No.
provider	Specifies the language provider of the processor, such as the Event Processor Language (EPL). Valid values are: • `epl` The default value is `epl`.	String	No.

Table 2-6 Attributes of the wlevs:processor Spring Tag

Attribute	Description	Data Type	Required?
queryURL	Specifies a URL that points to an EPL rules definition file for this processor.	String.	No.
manageable	Specifies that this component can be monitored using the WebLogic Event Server monitoring framework. Setting this attribute to `true` may adversely impact the performance of your application, so change the default setting of this attribute only if you are sure you want monitoring information about this component. Valid values for this attribute are `true` and `false`. The default value is `false`.	Boolean	No.

Example

The following example shows how to use the `wlevs:processor` tag in the EPN assembly file:

```
<wlevs:processor id="spreader" />
```

The example shows how to declare a processor with ID `spreader`. This means that in the processor configuration file that contains the EPL rules for this processor, the `<name>` element must contain the value `spreader`. This way WebLogic Event Server knows which EPL rules it must file for this particular processor.

wlevs:property

Specifies a custom property to apply to the event type.

This tag is used only as a child of wlevs:event-type, wlevs:adapter, wlevs:processor, or wlevs:stream.

The `wlevs:property` tag is defined as the Spring `propertyType` type; for additional details of this Spring data type, the definition of the allowed child tags, and so on, see the Spring 2.0 XSD.

Child Tags

You can specify one of the following standard Spring tags as a child element of the `wlevs:property` tag:

- `meta`

- bean
- ref
- idref
- value
- null
- list
- set
- map
- props

Attributes

The following table lists the attributes of the wlevs:property Spring tag.

Table 2-7 Attributes of the wlevs:property Spring Tag

Attribute	Description	Data Type	Required?
name	Specifies the name of the property, following JavaBean naming conventions.	String	Yes.
ref	A short-cut alternative to a nested <ref bean='...'/> element.	String	No.
value	A short-cut alternative to a nested <value>...</value> element.	String	No.

Example

The following example shows how to use the wlevs:property tag in the EPN assembly file:

```
<wlevs:event-type type-name="ForeignExchangeEvent">
   <wlevs:metadata>
      <entry key="symbol" value="java.lang.String"/>
      <entry key="price" value="java.lang.Double"/>
   </wlevs:metadata>
```

```
    <wlevs:property name="builderFactory">
      <bean id="builderFactory"
            class="com.bea.wlevs.example.fx.ForeignExchangeBuilderFactory"/>
    </wlevs:property>
</wlevs:event-type>
```

In the example, the `wlevs:property` tag defines a custom property of the `ForeignExchangeEvent` called `builderFactory`. The property uses the standard Spring `bean` tag to specify the Spring bean used as a factory to create `ForeignExchangeEvents`.

wlevs:source

Specifies an event source for this stream, or in other words, the component which this stream is coming *from*. Specifying an event source is equivalent to specifying this stream as an event listener to another component.

You can also nest the definition of a component within a particular `wlevs:source` component to specify the component source.

WARNING: Nested definitions are not eligible for dynamic configuration or monitoring.

This tag is only a child of wlevs:stream.

Attributes

The following table lists the attributes of the `wlevs:source` Spring tag.

Table 2-8 Attributes of the wlevs:source Spring Tag

Attribute	Description	Data Type	Required?
ref	Specifies the source of the stream to which this tag is a child. Set this attribute to the value of the `id` attribute of the source component. You do not specify this attribute if you are nesting sources.	String	No.

Example

The following example shows how to use the `wlevs:source` tag in the EPN assembly file:

```
<wlevs:stream id="helloworldInstream">
    <wlevs:listener ref="helloworldProcessor"/>
    <wlevs:source ref="helloworldAdapter"/>
</wlevs:stream>
```

In the example, the component with `id helloworldAdapter` is the source of the `helloworldInstream` stream component.

wlevs:stream

Use this tag to declare a stream to the Spring application context.

Child Tags

The `wlevs:stream` Spring tag supports the following child tags:

- wlevs:listener
- wlevs:source
- wlevs:instance-property
- wlevs:property

Attributes

The following table lists the attributes of the `wlevs:stream` Spring tag.

Table 2-9 Attributes of the wlevs:stream Spring Tag

Attribute	Description	Data Type	Required?
id	Unique identifier for this component.	String	Yes.
advertise	Advertises this service in the OSGi registry. Valid values are `true` and `false`. Default value is `false`.	Boolean	No.

Table 2-9 Attributes of the wlevs:stream Spring Tag

Attribute	Description	Data Type	Required?
depends-on	The name of the Spring beans that the underlying Spring bean that implements this componet depends on. Use this attribute to enforce the correct initialization order of beans in your application. If you do not require any specific initialization order, then do not specify this attribute. If using, set this attribute to the value of the `id` attribute of the dependent Spring bean.	String	No.
lazy-init	Specifies whether WeblogicEvent Server should lazily initialize the underlying Spring bean that implements this component. If set to `false`, the bean will be instantiated on startup by bean factories that perform eager initialization of singletons. Valid values for this attribute are `true` and `false`. The default value is `false`.	Boolean.	No.
listeners	Specifies the components that listen to this component. Separate multiple components using commas. Set this attribute to the value of the `id` attribute of the tag (`wlevs:adapter`, `wlevs:stream`, or `wlevs:processor`) that defines the listening component.	String	No.
provider	Specifies the streaming provider. Valid values are: • `defaultstream` Default value is `defaultstream`, which is the out-of-the-box streaming provider.	String	No.
manageable	Specifies that this component can be monitored using the WebLogic Event Server monitoring framework. Setting this attribute to `true` may adversely impact the performance of your application, so change the default setting of this attribute only if you are sure you want monitoring information about this component. Valid values for this attribute are `true` and `false`. The default value is `false`..	Boolean	No.

Table 2-9 Attributes of the wlevs:stream Spring Tag

Attribute	Description	Data Type	Required?
max-size	Specifies the maximum size of this stream. Zero-size streams synchronously pass-through events. Streams with non-zero size process events asynchronously, buffering events by the requested size. The default value for this attribute is `1024`.	integer	No.
max-threads	Specifies the maximum number of threads that will be used to process events for this stream. If the max-size attribute is 0, then setting a value for max-threads has no effect. The default value for this attribute is 1.	integer	No.
source	Specifies the component from which the stream sources events. Set this attribute to the value of the `id` attribute of the tag (`wlevs:adapter`, `wlevs:stream`, or `wlevs:processor`) that defines the source component.	String	No.

Example

The following example shows how to use the `wlevs:stream` tag in the EPN assembly file:

```
<wlevs:stream id="fxMarketAmerOut" />
```

The example shows how to declare a stream service with unique identifier `fxMarketAmerOut`.

CHAPTER 3

Deployer Command-Line Reference

This section contains information on the following subjects:

Overview of Using the Deployer Command-Line Utility

`com.bea.wlevs.deployment.Deployer` is a Java-based deployment utility that provides administrators and developers command-line based operations for deploying WebLogic Event Server applications. In the context of WebLogic Event Server deployment, an application is defined as an OSGi bundle JAR file that contains the following artifacts:

- The compiled Java class files that implement some of the components of the application, such as the adapters, adapter factory, and POJO that contains the business logic. T
- One or more WebLogic Event Server configuration XML files that configure the components of the application, such as the processor, adapter, or streams.

  The configuration files must be located in the `META-INF/wlevs` directory of the OSGi bundle JAR file.

- An EPN assembly file that describes all the components of the application and how they are connected to each other. The EPN assembly file extends the standard Spring context file.

  The EPN assembly file must be located in the `META-INF/spring` directory of the OSGi bundle JAR file.

- A `MANIFEST.MF` file that describes the contents of the JAR.

See Assembling a WebLogic Event Server Application: Main Steps for detailed instructions on creating this deployment bundle.

The Deployer utility uses HTTP to connect to WebLogic Event Server, which means that you must configure Jetty for the server instance to which you are deploying your application. For details, see Configuring WebLogic Event Server.

WebLogic Event Server uses the `deployments.xml` file to internally maintain its list of deployed application OSGi bundles. This file is located in in the *`DOMAIN_DIR`* directory, where *`DOMAIN_DIR`* refers to the main domain directory correspoding to the server instance to which you are deploying your application. See "Deployment XSD Schema" on page 5-4 for information about this file.

WARNING: The XSD for the `deployments.xml` file is provided for your information only; BEA does not recommend updating the `deployments.xml` file manually.

Required Environment for the Deployer Utility

To set up your environment to use the `com.bea.wlevs.deployment.Deployer` utility:

1. Install and configure the WebLogic Event Server software, as described in Installing WebLogic Event Server.
2. Open a command window and set your environment as described in Setting Up Your Development Environment.
3. Update your CLASSPATH variable to include the `com.bea.wlevs.deployment.client_2.0.jar JAR` file, located in the *`WLEVS_HOME`*`/bin` directory where, *`WLEVS_HOME`* refers to the main WebLogic Event Server installation directory, such as `/beahome/wlevs20`.

   Alternatively, you can later use the `-jar` option at the command line to call this JAR file, such as:

   ```
   prompt> java -jar
   /beahome/wlevs20/bin/com.bea.wlevs.deployment.client_2.0.jar -url ...
   ```

It is assumed in the remainder of this section that you have updated your CLASSPATH and are going to call the `com.bea.wlevs.deployment.Deployer` class directly.

Running the Deployer Utility Remotely

Sometimes it is useful to run the `com.bea.wlevs.deployment.Deployer` utility on a computer different from the computer on which WebLogic Event Server is installed and running. To run the utility remotely, follow these steps:

1. Copy the following JAR files from the computer on which WebLogic Event Server is installed to the computer on which you want to run the deployer utility; you can copy the JAR files to the directory name of your choice:
   - *BEA_HOME*`/wlevs20/bin/com.bea.wlevs.deployment.client_2.0.jar`

   where *BEA_HOME* refers to the main BEA Home directory into which you installed WebLogic Event Server.
2. Set your CLASSPATH in one of the following ways:
   - Implicitly set your CLASSPATH by using the `-jar` argument when you run the utility; set the argument to the *NEW_DIRECTORY*`/com.bea.wlevs.deployment.client_2.0.jar` file, where *NEW_DIRECTORY* refers to the directory on the remote computer into which you copied the required JAR file. When you use the `-jar` argument, you do not specify the `com.bea.wlevs.deployment.Deployer` utility name at the command line.
   - Explicitly update your CLASSPATH by adding the JAR file you copied to the remote computer to your CLASSPATH environment variable:
3. Invoke the `com.bea.wlevs.deployment.Deployer` utility as described in the next section.

Syntax for Invoking the Deployer Utility

The syntax for using the Deployer utility is as follows:

```
java  com.bea.wlevs.deployment.Deployer
      [Connection Arguments]
      [User Credential Arguments]
      [Deployment Commands]
```

The following sections describe the various arguments and commands you can use with the Deployer utility. See "Examples of Using the Deployer Utility" on page 3-7 for specific examples of using the utility.

Connection Arguments

The following table describes the connection arguments you can specify with the Deployer utility.

Table 3-1 Connection Arguments

Argument	Description

`-url` *`url`*	Specifies the URL of the deployer of the WebLogic Event Server instance to which you want to deploy the OSGI bundle.
	The URL takes the following form:
	`http://`*`host`*`:`*`port`*`/wlevsdeployer`
	where:
	• *`host`* refers to the hostname of the computer on which WebLogic Event Server is running.
	• *`port`* refers to the port number to which WebLogic Event Server listens; its value is `9002` by default. This port is specified in the `config.xml` file that describes your WebLogic Event Server domain, located in the *`DOMAIN_DIR`*`/config` directory, where *`DOMAIN_DIR`* refers to your domain directory. The port number is the value of the `<Port>` child element of the `<Netio>` element:
	`<Netio>`
	`  <Name>NetIO</Name>`
	`  <Port>9002</Port>`
	`</Netio>`
	For example, if WebLogic Event Server is running on host `ariel` at port `9002`, then the URL would be:
	`http://ariel:9002/wlevsdeployer`

User Credential Arguments

The following table describes the user credential arguments you can specify with the Deployer utility.

Table 3-2 User Credential Arguments

Argument	Description
`-user` *`username`*	Username of the WebLogic Event Server administrator. If you supply the `-user` option but you do not supply a corresponding `-password` option, the Deployer utility prompts you for the password.
`-password` *`password`*	Password of the WebLogic Event Server administrator.

Deployment Commands

The following table describes the deployment commands you can specify with the Deployer utility.

Table 3-3 Deployment Commands

Command	Description
`-install` *`bundle`*	Installs the specified OSGi bundle to the specified WebLogic Event Server instance. Be sure to specify the full pathname of the bundle if it is not located relative to the directory from which you are running the Deployer utility. In particular, WebLogic Event Server: • Copies the specified bundle to the domain directory. • Searches the `META-INF/wlevs` directory in the bundle for the component configuration files and extracts them to the domain directory. • Updates the internal deployment registry.
`-update` *`bundle`*	Updates the existing OSGi bundle with new application code. Be sure to specify the full pathname of the bundle if it is not located relative to the directory from which you are running the Deployer utility. In particular, WebLogic Event Server: • Copies the updated bundles to the domain directory. • Searches the `META-INF/wlevs` directory in the updated bundle for the updated component configuration files and extracts them to the domain directory. • Updates the internal deployment registry with the updated information.
`-uninstall` *`name`*	Removes the existing bundle from the specified WebLogic Event Server instance. The *`name`* parameter refers to the symbolic name of the OSGi bundle that you want to remove. The symbolic name is the value of the `Bundle-SymbolicName` header in the bundle's `MANIFEST.MF` file: `Bundle-SymbolicName: myApp` In particular, WebLogic Event Server: • Removes the specified OSGi bundle from the domain directory. • Removes the bundles from the internal deployment registry .

Table 3-3 Deployment Commands

Command	Description
`-start` *`name`*	Starts an OSGI bundle which was previously installed to the specified WebLogic Event Server instance. The *`name`* parameter refers to the symbolic name of the OSGi bundle that you want to start. The symbolic name is the value of the `Bundle-SymbolicName` header in the bundle's `MANIFEST.MF` file: `Bundle-SymbolicName: myApp`
`-stop` *`name`*	Stops an OSGI bundle that is currently running on the specified WebLogic Event Server instance. The *`name`* parameter refers to the symbolic name of the OSGi bundle that you want to stop. The symbolic name is the value of the `Bundle-SymbolicName` header in the bundle's `MANIFEST.MF` file: `Bundle-SymbolicName: myApp`
`-status` *`name`*	Returns status information about a currently installed OSGi bundle. The *`name`* parameter refers to the symbolic name of the OSGi bundle for which you want status information. The symbolic name is the value of the `Bundle-SymbolicName` header in the bundle's `MANIFEST.MF` file: `Bundle-SymbolicName: myApp`
`-startLevel` *`startLevel`*	Specifies the level at which the OSGi bundle is started. Bundles with smaller numbers are started first. System bundles have start levels of under 7.

Examples of Using the Deployer Utility

The following examples show how to use the Deployer utility. In all the examples, WebLogic Event Server is running on host `ariel`, listening at port `9002`, and the username/password of the server administrator is `wlevs/wlevs`, respectively. For clarity, the examples are shown on multiple lines; however, when you run the command, enter all arguments and commands on a single line.

```
prompt> java com.bea.wlevs.deployment.Deployer
        -url http://ariel:9002/wlevsdeployer -user wlevs -password wlevs
         -install /application/bundles/com.my.exampleApp_1.0.0.0.jar
```

The preceding example shows how to install an OSGi bundle called `com.my.exampleApp_1.0.0.0.jar`, located in the `/application/bundles` directory. The next command shows how to start this application after it has been installed:

```
prompt> java com.bea.wlevs.deployment.Deployer
        -url http://ariel:9002/wlevsdeployer -user wlevs -password wlevs
         -start exampleApp
```

Finally, the next example shows how to uninstall the application, which removes all traces of it from the domain directory:

```
prompt> java com.bea.wlevs.deployment.Deployer
        -url http://ariel:9002/wlevsdeployer -user wlevs -password wlevs
         -uninstall exampleApp
```

CHAPTER 4

Metadata Annotations

This section contains information on the following subjects:

Overview of WebLogic Event Server Metadata Annotations

The WebLogic Event Server metadata annotations are used to access the configuration of a component.

You use the following three annotations to specify the methods of an adapter Java implementation that handle various stages of the adapter's lifecyle: when its configuration is prepared, when the configuration is activated, and when the adapter is terminated due to an exception:

- com.bea.wlevs.management.Activate
- com.bea.wlevs.management.Prepare
- com.bea.wlevs.management.Rollback

Use the com.bea.wlevs.util.Service annotation to specify the method of a component that is injected with an OSGi service reference.

com.bea.wlevs.management.Activate

Target: Method

The `@Activate` annotation is one of the adapter lifecycle annotations that you use in the Java file that implements your custom adapter to explicitly specify the methods that WebLogic Event Server uses to send configuration information to the adapter.

WebLogic Event Server calls methods marked with the `@Activate` annotation after, and if, the server has called and successfully executed all the methods marked with the `@Prepare` annotation. You typically use the `@Activate` method to actually get the adapter's configuration data to use in the rest of the adapter implementation.

The method you annotate with this annotation must have the following signature:

```
public void methodName(AdapterConfigObject adapterConfig)
```

where *`methodName`* refers to the name you give the method and *`AdapterConfigObject`* refers to the Java represenation of the adapter's configuration XML file which is deployed with the application. The type of this class is `com.bea.wlevs.configuration.application.DefaultAdapterConfig` by default; if, however, you have extended the configuration of the adapter, then the type is whatever have specified in the XSD that describes the extended XML file. For example, in the HelloWorld sample, the type is `com.bea.wlevs.adapter.example.helloworld.HelloWorldAdapterConfig`.

At run time, WebLogic Event Server automatically creates an instance of this class, populates it with data from the actual XML file, and passes the instance to the adapter. The adapter methods annotated with the adapter lifecycle annotations can then use the class to get information about the adapter's configuration.

This metadata annotation does not have any attributes.

Example

The following sample code from the adapter component of the HelloWorld example shows how to use the `@Prepare` annotation; only relevant code is shown:

```
package com.bea.wlevs.adapter.example.helloworld;
```

```
...

import com.bea.wlevs.configuration.Activate;
import com.bea.wlevs.configuration.Prepare;
import com.bea.wlevs.configuration.Rollback;
import com.bea.wlevs.ede.api.Adapter;
import com.bea.wlevs.ede.api.EventSender;
import com.bea.wlevs.ede.api.EventSource;
import com.bea.wlevs.ede.api.SuspendableBean;

import com.bea.wlevs.adapter.example.helloworld.HelloWorldAdapterConfig;

public class HelloWorldAdapter implements Runnable, Adapter, EventSource,
SuspendableBean {

...

    @Activate
    public void activateAdapter(HelloWorldAdapterConfig adapterConfig) {
        this.message = adapterConfig.getMessage();
    }

...

}
```

The `com.bea.wlevs.adapter.example.helloworld.HelloWorldAdapterConfig` class is a Java represenation of the adapter's configuration XML file. In the HelloWorld example, the configuration has been extended; this means a custom XSD file describes the XML file. The following XSD file also specifies the fully qualified name of the resulting Java configuration object, as shown in bold:

```
<?xml version="1.0" encoding="UTF-8"?>

<xs:schema xmlns="http://www.bea.com/ns/wlevs/example/helloworld"
 xmlns:xs="http://www.w3.org/2001/XMLSchema"
 xmlns:jxb="http://java.sun.com/xml/ns/jaxb"
 xmlns:xjc="http://java.sun.com/xml/ns/jaxb/xjc"
 xmlns:wlevs="http://www.bea.com/ns/wlevs/config/application"
 targetNamespace="http://www.bea.com/ns/wlevs/example/helloworld"
 elementFormDefault="unqualified" attributeFormDefault="unqualified"
 jxb:extensionBindingPrefixes="xjc" jxb:version="1.0">

   <xs:annotation>
      <xs:appinfo>
        <jxb:schemaBindings>
          <jxb:package name="com.bea.wlevs.adapter.example.helloworld"/>
        </jxb:schemaBindings>
      </xs:appinfo>
   </xs:annotation>
```

```
   <xs:import namespace="http://www.bea.com/ns/wlevs/config/application"
schemaLocation="wlevs_application_config.xsd"/>

   <xs:element name="config">
     <xs:complexType>
       <xs:choice maxOccurs="unbounded">
          <xs:element name="adapter" type="HelloWorldAdapterConfig"/>
          <xs:element name="processor" type="wlevs:DefaultProcessorConfig"/>
          <xs:element name="stream" type="wlevs:DefaultStreamConfig" />
       </xs:choice>
     </xs:complexType>
    </xs:element>

    <xs:complexType name="HelloWorldAdapterConfig">
      <xs:complexContent>
        <xs:extension base="wlevs:AdapterConfig">
          <xs:sequence>
            <xs:element name="message" type="xs:string"/>
          </xs:sequence>
        </xs:extension>
      </xs:complexContent>
    </xs:complexType>

</xs:schema>
```

WebLogic Event Server automatically creates an instance of this class when the application is deployed. For example, the adapter section of the `helloworldAdapter's` configuration file is as follows:

```
<?xml version="1.0" encoding="UTF-8"?>

<helloworld:config

...

  <adapter>
    <name>helloworldAdapter</name>
    <message>HelloWorld - the current time is:</message>
  </adapter>
</helloworld:config>
```

In the Java code of the adapter above, the `activateAdapter` method is annotated with the `@Activate` annotation. The method uses the `getMessage` method of the configuration object to get the value of the `message` property set in the adapter's configuration XML file. In this case, the value is `HelloWorld - the current time is:`. This value can then be used in the main part of the adapter implementation file.

com.bea.wlevs.management.Prepare

Target: Method

The `@Prepare` annotation is one of the adapter lifecycle annotations that you use in the Java file that implements your custom adapter to explicitly specify the methods that WebLogic Event Server uses to send configuration information to the adapter.

WebLogic Event Server calls the method annotated with `@Prepare` whenever a component's state has been updated by a particular configuration change.

The method you annotate with this annotation must have the following signature:

```
public void methodName(AdapterConfigObject adapterConfig)
```

where *`methodName`* refers to the name you give the method and *`AdapterConfigObject`* refers to the Java represenation of the adapter's configuration XML file which is deployed with the application. The type of this class is `com.bea.wlevs.configuration.application.DefaultAdapterConfig` by default; if, however, you have extended the configuration of the adapter, then the type is whatever have specified in the XSD that describes the extended XML file. For example, in the HelloWorld sample, the type is `com.bea.wlevs.adapter.example.helloworld.HelloWorldAdapterConfig`.

At run time, WebLogic Event Server automatically creates an instance of this class, populates it with data from the actual XML file, and passes the instance to the adapter. The adapter methods annotated with the adapter lifecycle annotations can then use the class to get information about the adapter's configuration.

This metadata annotation does not have any attributes.

Example

The following sample code from the adapter component of the HelloWorld example shows how to use the `@Prepare` annotation; only relevant code is shown:

```
package com.bea.wlevs.adapter.example.helloworld;

...

import com.bea.wlevs.configuration.Activate;
import com.bea.wlevs.configuration.Prepare;
import com.bea.wlevs.configuration.Rollback;
import com.bea.wlevs.ede.api.Adapter;
import com.bea.wlevs.ede.api.EventSender;
```

```
import com.bea.wlevs.ede.api.EventSource;
import com.bea.wlevs.ede.api.SuspendableBean;

import com.bea.wlevs.adapter.example.helloworld.HelloWorldAdapterConfig;

public class HelloWorldAdapter implements Runnable, Adapter, EventSource,
SuspendableBean {

...

    @Prepare
    public void checkConfiguration(HelloWorldAdapterConfig adapterConfig) {
        if (adapterConfig.getMessage() == null
                || adapterConfig.getMessage().length() == 0) {
            throw new RuntimeException("invalid message: " + message);
        }
    }

...

}
```

The `com.bea.wlevs.adapter.example.helloworld.HelloWorldAdapterConfig` class is a Java represenation of the adapter's configuration XML file; WebLogic Event Server automatically creates an instance of this class when the application is deployed. In the HelloWorld example, the adapter configuration has been extended. See the example in "com.bea.wlevs.management.Activate" on page 4-2 for additional details.

In the Java code of the adapter above, the `checkConfiguration` method is annotated with the `@Prepare` annotation, which means this method is called when the adapter's configuration changes in some way. The example further shows that the method checks to make sure that the `message` property of the adapter's configuration (set in the extended adapter configuration file) is not null or empty; if it is, then the method throws an exception.

com.bea.wlevs.management.Rollback

Target: Method

The `@Rollback` annotation is one of the adapter lifecycle annotations that you use in the Java file that implements your custom adapter to explicitly specify the methods that WebLogic Event Server uses to send configuration information to the adapter.

WebLogic Event Server calls the method annotated with `@Rollback` whenever a component whose `@Prepare` method was called but threw an exception. The server calls the `@Rollback` method for each component for which this is true.

The method you annotate with this annotation must have the following signature:

```
public void methodName(AdapterConfigObject adapterConfig)
```

where *methodName* refers to the name you give the method and *AdapterConfigObject* refers to the Java represenation of the adapter's configuration XML file which is deployed with the application. The type of this class is `com.bea.wlevs.configuration.application.DefaultAdapterConfig` by default; if, however, you have extended the configuration of the adapter, then the type is whatever have specified in the XSD that describes the extended XML file. For example, in the HelloWorld sample, the type is `com.bea.wlevs.adapter.example.helloworld.HelloWorldAdapterConfig`.

At run time, WebLogic Event Server automatically creates an instance of this class, populates it with data from the actual XML file, and passes the instance to the adapter. The adapter methods annotated with the adapter lifecycle annotations can then use the class to get information about the adapter's configuration.

This metadata annotation does not have any attributes.

Example

The following sample code from the adapter component of the HelloWorld example shows how to use the `@Rollback` annotation; only relevant code is shown:

```
package com.bea.wlevs.adapter.example.helloworld;

...

import com.bea.wlevs.configuration.Activate;
import com.bea.wlevs.configuration.Prepare;
import com.bea.wlevs.configuration.Rollback;
import com.bea.wlevs.ede.api.Adapter;
import com.bea.wlevs.ede.api.EventSender;
import com.bea.wlevs.ede.api.EventSource;
import com.bea.wlevs.ede.api.SuspendableBean;

import com.bea.wlevs.adapter.example.helloworld.HelloWorldAdapterConfig;

public class HelloWorldAdapter implements Runnable, Adapter, EventSource, Suspen

dableBean {

...

    @Rollback
   public void rejectConfigurationChange(HelloWorldAdapterConfig adapterConfig)
{
    }
```

The `com.bea.wlevs.adapter.example.helloworld.HelloWorldAdapterConfig` class is a Java represenation of the adapter's configuration XML file; WebLogic Event Server automatically creates an instance of this class when the application is deployed. In the HelloWorld example, the adapter configuration has been extended. See the example in "com.bea.wlevs.management.Activate" on page 4-2 for additional details.

In the example, the `rejectConfigurationChange` method is annotated with the `@Rollback` annotation, which means this is the method that is called if the `@Prepare` method threw an exception. In the example above, nothing actually happens.

com.bea.wlevs.util.Service

Target: Method

Specifies that the annotated method, typically a JavaBean setter method, requires an OSGi service reference.

Attributes

Table 0-1 Attributes of the com.bea.wlevs.util.Service JWS Annotation Tag

Name	Description	Data Type	Required?
serviceBeanName	The name of the bean that backs the injected service. May be null.	String	No.
cardinality	Valid values for this attribute are: • `ServiceCardinality.C0__1` • `ServiceCardinality.C0__N` • `ServiceCardinality.C1__1` • `ServiceCardinality.C1__N` The default value is `ServiceCardinality.C1__1`.	enum	No.
contextClassloader	Valid values for this attribute are: • `ServiceClassloader.CLIENT` • `ServiceClassloader.SERVICE_PROVIDER` • `ServiceClassloader.UNMANAGED` The default value is `ServiceClassloader.CLIENT`.	enum	No.

Table 0-1 Attributes of the com.bea.wlevs.util.Service JWS Annotation Tag

Name	Description	Data Type	Required?
timeout	Timeout for servcie resolution in milliseconds. Default value is 30000.	int	No.
serviceType	Interface (or class) of the service to be injected Default value is `Service.class`.	Class	No.
filter	Specifies the filter used to narrow service matches. Value may be `null`.	String	No.

Example

The following example shows how to use the `@Service` annotation:

```
@Service(filter = "(Name=StockDs)")
public void setDataSourceService(DataSourceService dss) {
    initStockTable(dss.getDataSource());
}
```

CHAPTER 5

XSD Schema Reference for WebLogic Event Server Files

This section contains information on the following subjects:

Component Configuration XSD Schemas

The following XSD schema files describe the structure of the XML files you use to configure WebLogic Event Server components, such as the complex event processors and adapters. The `wlevs_application_config.xsd` schema imports the `wlevs_base_config.xsd` schema..

- wlevs_application_config.xsd
- wlevs_base_config.xsd

Example of a Component Configuration File

The following example shows the component configuration file for the HelloWorld sample application:

```
<?xml version="1.0" encoding="UTF-8"?>

<helloworld:config
  xmlns:helloworld="http://www.bea.com/ns/wlevs/example/helloworld">
  <processor>
```

```
    <name>helloworldProcessor</name>
    <rules>
      <rule id="helloworldRule"><![CDATA[ select * from HelloWorldEvent retain
1 event ]]></rule>
    </rules>
  </processor>

  <adapter>
    <name>helloworldAdapter</name>
    <message>HelloWorld - the current time is:</message>
  </adapter>

  <stream monitoring="true" >
      <name>helloworldOutstream</name>
      <max-size>10000</max-size>
      <max-threads>2</max-threads>
  </stream>

</helloworld:config>
```

EPN Assembly XSD Schema

You use the EPN assembly file to declare the components that make up your WebLogic Event Server application and how they are connected to each other, or in other words, the *event processing network*. The EPN assembly file is an extension of the standard Spring context file. You also use the file to register the Java classes that implement the adapter and POJO components of your application, register the event types that you use throughout your application and EPL rules, and reference in your environment the WebLogic Event Server-specific services.

See spring-wlevs.xsd for the full XSD Schema.

Example of a EPN Assembly File

The following XML file shows the EPN assembly file for the HelloWorld example:

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xmlns:osgi="http://www.springframework.org/schema/osgi"
       xmlns:wlevs="http://www.bea.com/ns/wlevs/spring"
       xsi:schemaLocation="
  http://www.springframework.org/schema/beans
  http://www.springframework.org/schema/beans/spring-beans.xsd
  http://www.springframework.org/schema/osgi
  http://www.springframework.org/schema/osgi/spring-osgi.xsd
```

```
  http://www.bea.com/ns/wlevs/spring
  http://www.bea.com/ns/wlevs/spring/spring-wlevs.xsd">

    <!-- First, create and register the adapter (and factory) that generates
hello world messages -->
    <osgi:service interface="com.bea.wlevs.ede.api.AdapterFactory">
        <osgi:service-properties>
            <prop key="type">hellomsgs</prop>
        </osgi:service-properties>
        <bean
class="com.bea.wlevs.adapter.example.helloworld.HelloWorldAdapterFactory" />
    </osgi:service>

    <wlevs:event-type-repository>
        <wlevs:event-type type-name="HelloWorldEvent">

<wlevs:class>com.bea.wlevs.event.example.helloworld.HelloWorldEvent</wlevs:cla
ss>
        </wlevs:event-type>
    </wlevs:event-type-repository>

    <!-- Assemble EPN (event processing network) -->

    <!-- The adapter id is used by the configuration system, so needs to be
well-known -->

    <wlevs:adapter id="helloworldAdapter" provider="hellomsgs"
manageable="true">
        <!-- This property is also configure by dynamic config -->
       <wlevs:instance-property name="message" value="HelloWorld - the current
time is:"/>
    </wlevs:adapter>

    <!-- The processor id is used by the configuration system, so needs to be
well-known -->

    <wlevs:processor id="helloworldProcessor" manageable="true" />

    <wlevs:stream id="helloworldInstream" manageable="true">
        <wlevs:listener ref="helloworldProcessor"/>
        <wlevs:source ref="helloworldAdapter"/>
    </wlevs:stream>

    <!-- Manageable is so that we can monitor the event throughput -->

    <wlevs:stream id="helloworldOutstream" manageable="true">
        <wlevs:listener>
            <!-- Create business object -->
            <bean class="com.bea.wlevs.example.helloworld.HelloWorldBean"/>
        </wlevs:listener>
```

```
        <wlevs:source ref="helloworldProcessor"/>
    </wlevs:stream>

</beans>
```

Deployment XSD Schema

The deployment file for WebLogic Event Server is called `deployments.xml` and is located in the *DOMAIN_DIR* directory, where *DOMAIN_DIR* refers to the main domain directory. This XML file lists the OSGi bundles that have been deployed to the server.

See deployment..xsd for the full XSD Schema.

Example of a Deployment XML File

The following example shows the `deployments.xml` file for the sample FX domain:

```
<?xml version="1.0" encoding="UTF-8"?>

<beans xmlns="http://www.springframework.org/schema/beans"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns:wlevs="http://www.bea.com/ns/wlevs/deployment"
  xsi:schemaLocation="
  http://www.springframework.org/schema/beans
  http://www.springframework.org/schema/beans/spring-beans.xsd
  http://www.bea.com/ns/wlevs/deployment
  http://www.bea.com/ns/wlevs/deployment/deployment.xsd">

  <wlevs:deployment id="fxApp" state="start"
location="file:applications/fx/com.bea.wlevs.example.fx_2.0.0.0.jar"/>

</beans>
```