Oracle® Complex Event Processing

Reference Guide

Release 10*g*R3 (10.3)

September 2008

ORACLE

Oracle Complex Event Processing Reference Guide, Release 10*g*R3 (10.3)

Contents

1. Introduction and Roadmap

2. Oracle CEP Application Assembly Tag Reference

CHAPTER 1

Introduction and Roadmap

This section describes the contents and organization of this guide—*Oracle Complex Event Processing Reference Guide*.

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 Oracle Complex Event Processing (or *Oracle CEP* for short)or considering the use of Oracle CEP 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.

Oracle CEP Documentation Set

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

- *Oracle CEP Getting Started*
- *Oracle CEP Application Development Guide*
- *Oracle CEP Administration and Configuration Guide*
- *Oracle CEP EPL Reference Guide*
- *Oracle CEP Reference Guide*
- *Oracle CEP Release Notes*
- *Oracle CEP Visualizer Help*
- *Oracle CEP Type 4 JDBC Drivers*

See the main Oracle CEP 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 Oracle CEP documentation set and samples.
- Chapter 2, "Oracle CEP Application Assembly Tag Reference," lists the Oracle CEP application assembly tags you can use in the EPN assembly file.
- Chapter 3, "Deployer Command-Line Reference," provides reference information for the Deployer tool used to install, update, start and stop OSGi bundles to Oracle CEP.
- 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 Oracle CEP Files," provides the XSD Schemas for the various Oracle CEP XML files, including component configuration files, EPN assembly file, deployments file, and server configuration file.

Samples for the Oracle CEP Application Developer

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

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

Note: When you initially install Oracle CEP, you must chose the `Custom` option to also install the examples. The `Typical` option does *not* include the examples.

If you previously installed Oracle CEP using the `Typical` option, and you now want to also install the examples, re-run the Oracle CEP installation process and specify the same Oracle CEP home directory; a later step in the installation process allows you to then install just the examples.

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 four examples are provided in both their own domain and as Java source in this release of Oracle CEP:

- HelloWorld—Example that shows the basic elements of an Oracle CEP application. See Hello World Example for additional information.

  The HelloWorld domain is located in *`ORACLE_CEP_HOME`*`\ocep_10.3\samples\domains\helloworld_domain`, where *`ORACLE_CEP_HOME`* refers to the Oracle CEP installation directory, such as `c:\oracle_cep`.

  The HelloWorld Java source code and configuration files are located in *`ORACLE_CEP_HOME`*`\ocep_10.3\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 Oracle CEP application. See Foreign Exchange (FX) Example for additional information.

  The ForeignExchange domain is located in *`ORACLE_CEP_HOME`*`\ocep_10.3\samples\domains\fx_domain`, where

ORACLE_CEP_HOME refers to the Oracle CEP installation directory, such as `c:\oracle_cep`.

The ForeignExchange Java source code and configuration files are located in *ORACLE_CEP_HOME*`\ocep_10.3\samples\source\applications\fx`.

- Signal Generation—Example that receives simulated market data and verifies if the price of a security has fluctuated more than two percent, and then detects if there is a *trend* occurring by keeping track of successive stock prices for a particular symbol.See Signal Generation Example for additional information.

  The Signal Generation domain is located in *ORACLE_CEP_HOME*`\ocep_10.3\samples\domains\signalgeneration_domain`, where *ORACLE_CEP_HOME* refers to the Oracle CEP installation directory, such as `c:\oracle_cep`.

  The Signal Generation Java source code and configuration files are located in *ORACLE_CEP_HOME*`\ocep_10.3\samples\source\applications\signalgeneration`.

- Record and Playback—Example that shows how to configure the recording and playback of events to a persistent event store, as well as how to use the built-in HTTP pub-sub adapter to publish messages to a channel. See Event Record and Playback Example for additional information.

  The Record and Playback domain is located in *ORACLE_CEP_HOME*`\ocep_10.3\samples\domains\recplay_domain`, where *ORACLE_CEP_HOME* refers to the Oracle CEP installation directory, such as `c:\oracle_cep`.

  The Record and Playback Java source code and configuration files are located in *ORACLE_CEP_HOME*`\ocep_10.3\samples\source\applications\recplay`.

CHAPTER 2

Oracle CEP Application Assembly Tag Reference

This section contains information on the following subjects:

Overview of the Oracle CEP Application Assembly Tags

Oracle Complex Event Processing, or *Oracle CEP* for short, provides a number of application assembly 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 Oracle CEP application assembly tags.

Figure 2-1 Hierarchy of Oracle CEP Application Assembly Tags

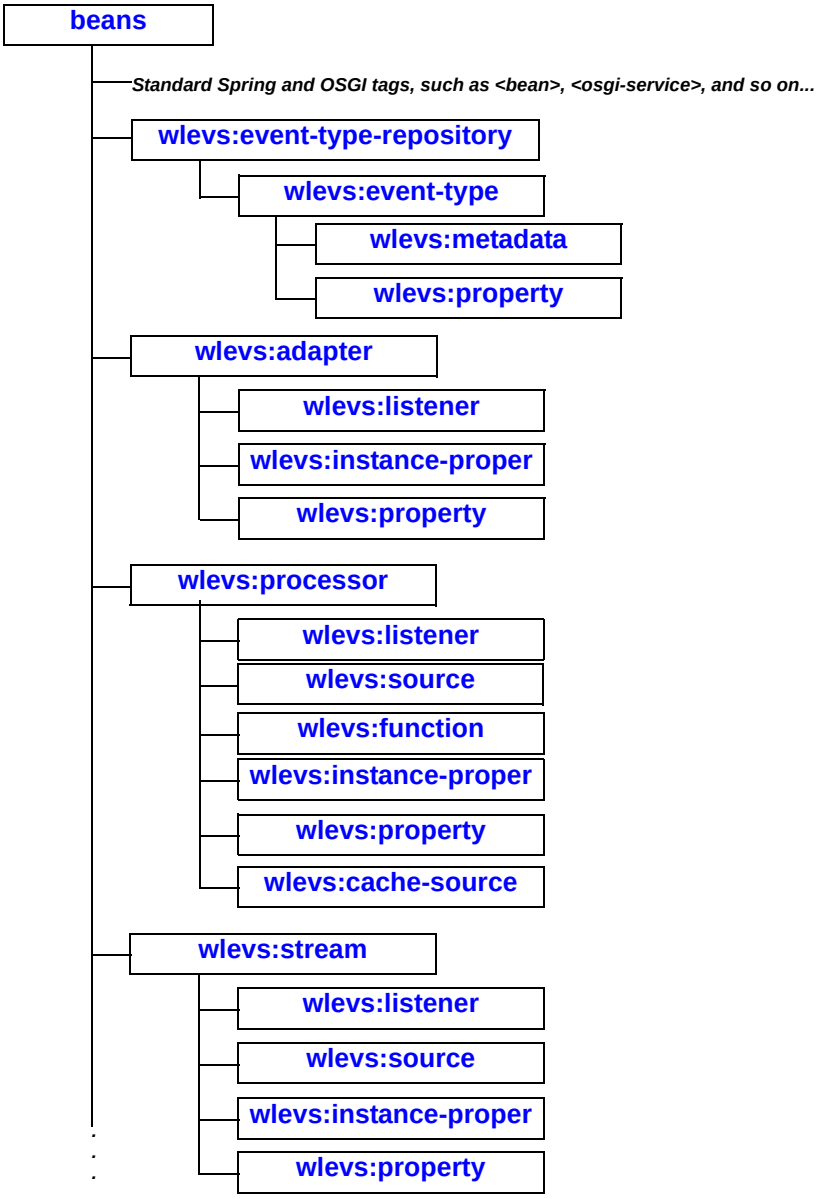
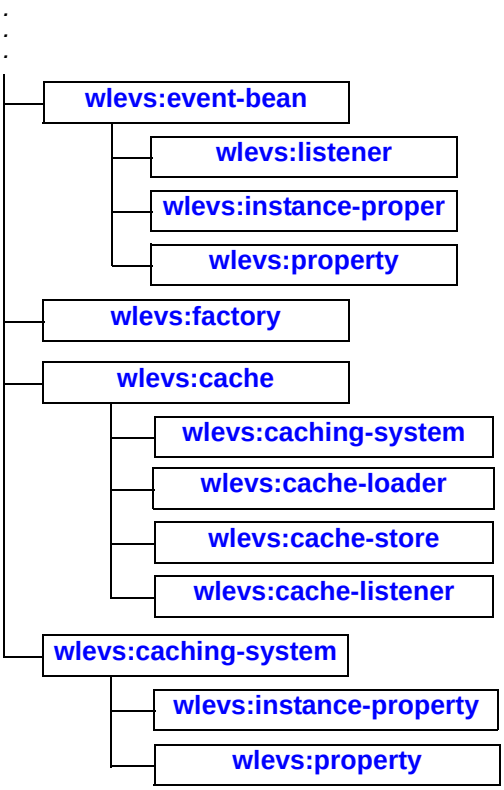


Figure 2-2 Hierarchy of Oracle CEP Application Assembly Tags (continued)



Example of an EPN Assembly File That Uses Oracle CEP Tags

The following sample EPN assembly file from the HelloWorld application shows how to use many of the Oracle CEP 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">

    <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>

    <wlevs:adapter id="helloworldAdapter"

class="com.bea.wlevs.adapter.example.helloworld.HelloWorldAdapter" >
        <wlevs:instance-property name="message"
                                 value="HelloWorld - the currenttime is:"/>
    </wlevs:adapter>

    <wlevs:processor id="helloworldProcessor" />

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

    <wlevs:stream id="helloworldOutstream" advertise="true">
        <wlevs:listener>
            <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 component to the Spring application context.

Child Tags

The `wlevs:adapter` application assembly tag supports the following child tags:

- wlevs:listener
- wlevs:instance-property

- wlevs:property

Attributes

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

Table 2-1 Attributes of the wlevs:adapter Application Assembly Tag

Attribute	Description	Data Type	Required?
id	Unique identifier for this component. This identifier must correspond to the `<name>` element in the XML configuration file for this adapter, if one exists.	String	Yes.
advertise	Advertises this service in the OSGi registry. Valid values are `true` and `false`. 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 Application Assembly 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"`
	If you are using one of the built-in HTTP publish-subscribe adapters, then specify the following hard-coded values:
	• For the built-in pub-sub adapter used for *publishing*, specify the hard-coded `httppub` value, such as:
	`provider="httppub"`
	• For the built-in pub-sub adapter used for *subscribing*, specify the hard-coded `httpsub` value, such as:
	`provider="httpsub"`
	If you are using a JMS adapter, then specify one of the following hard-coded values:
	• For the inbound JMS adapter, specify the `jms-inbound` value, such as:
	`provider="jms-inbound"`
	• For the outbound JMS adapter, specify the `jms-outbound` value, such as:
	`provider="jms-outbound"`
	You must specify either the `provider` or `class` attribute, but not both, otherwise an exception is raised.
		String	No.

class	Specifies the Java class that implements this adapter. You must specify either the `provider` or `class` attribute, but not both, otherwise an exception is raised.	String	No
onevent-method	Specifies the method of the adapter implementation that corresponds to the lifecycle `onEvent` method. Oracle CEP invokes this method when the adapter receives an event.	String	No

Table 2-1 Attributes of the wlevs:adapter Application Assembly Tag

Attribute	Description	Data Type	Required?
init-method	Specifies the method of the adapter implementation that corresponds to the lifecycle `init` method. Oracle CEP invokes this method after it has set all the supplied instance properties. This method allows the adapter instance to perform initialization only possible when all bean properties have been set and to throw an exception in the event of misconfiguration.	String	No
activate-method	Specifies the method of the adapter implementation that corresponds to the lifecycle `activate` method. Oracle CEP invokes this method after the dynamic configuration of the adapter has completed. This method allows the adapter instance to perform initialization only possible when all dynamic bean properties have been set and the EPN has been wired.	String	No
suspend-method	Specifies the method of the adapter implementation that corresponds to the lifecycle `suspend` method. Oracle CEP invokes this method when the application is suspended.	String	No
destroy-method	Specifies the method of the adapter implementation that corresponds to the lifecycle `destroy` method. Oracle CEP invokes this method when the application is stopped.	String	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:cache

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

Child Tags

The `wlevs:cache` application assembly tag supports the following child tags.

- `wlevs:caching-system`—Specifies the caching system to which this cache belongs.

  Note: This child tag is different from the wlevs:caching-system tag used to *declare* a caching system. The child tag of the `wlevs:cache` tag takes a single attribute, `ref`, that *references* the `id` attribute of a declared caching system.

- wlevs:cache-loader—Specifies the cache loader for this cache.
- wlevs:cache-store—Specifies a cache store for this cache.
- wlevs:cache-listener—Specifies a listener for this cache, or a component to which the cache sends events.

Attributes

The following table lists the attributes of the `wlevs:cache` application assembly tag.

Table 2-2 Attributes of the wlevs:cache Application Assembly Tag

Attribute	Description	Data Type	Required?
id	Unique identifier for this component. This identifier must correspond to the `<name>` element in the XML configuration file for this cache.	String	Yes.
name	Specifies an alternate name for this cache. If not specified, then the name of the cache is the same as its `id` attribute.	String	No.

Table 2-2 Attributes of the wlevs:cache Application Assembly Tag

Attribute	Description	Data Type	Required?
key-properties	Specifies a comma-separated list of names of the properties that together form the unique key value for the objects in the cache, or *cache key*. A cache key may be composed of a single property or multiple properties. When you configure a cache as a listener in an event processing network, Oracle CEP inserts events that reach the cache using the unique key value as a key. If you specify a key class using the `key-class` attribute, then this attribute is optional. If you specify neither `key-properties` nor `key-class`, then Oracle CEP uses the event object itself as both the key and value when it inserts the event object into the cache.	String	No.
key-class	Specifies the name of the Java class used for the cache key when the key is a composite key. If you do not specify the `key-properties` attribute, then all properties on the `key-class` are assumed to be key properties. If you specify neither `key-properties` nor `key-class`, then Oracle CEP uses the event object itself as both the key and value when it inserts the event object into the cache	String	No.
value-type	Specifies the type for the values contained in the cache. Must be a valid type name in the event type repository. This attribute is required only if the cache is referenced in an EPL query. This is because the query processor needs to know the type of events in the cache.	String	No.
caching-system	Specifies the caching system in which this cache is contained. The value of this attribute corresponds to the `id` attribute of the appropriate `wlevs:caching-system` element.	String	Yes.
advertise	Advertises this service in the OSGi registry. Valid values are `true` and `false`. Default value is `false`.	Boolean	No.

Example

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

```
<wlevs:cache id="cache-id" name="alternative-cache-name">
    <wlevs:caching-system ref="caching-system-id"/>
    <wlevs:cache-listener ref="tradeListener" />
</wlevs:cache>
```

In the example, the cache's unique identifier is `cache-id` and its alternate name is `alternative-cache-name`. The caching system to which the cache belongs has an `id` of `caching-system-id`. The cache has a listener to which the cache sends events; the component that listens to it has an `id` of `tradeListener`.

wlevs:cache-listener

Use this tag to specify a cache as a source of events to the listening component. The listening component must implement the `com.bea.cache.jcache.CacheListener` interface.

This tag is always a child of wlevs:cache.

Attributes

The following table lists the attributes of the `wlevs:cache-listener` application assembly tag.

Table 2-3 Attributes of the wlevs:cache-listener Application Assembly Tag

Attribute	Description	Data Type	Required?
ref	Specifies the component that listens to this cache. Set this attribute to the value of the `id` attribute of the listening component. The listening component can be an adapter or a Spring bean.	String	No.

Example

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

```
<wlevs:caching-system id="caching-system-id"/>
...
<wlevs:cache id="cache-id" name="alternative-cache-name">
    <wlevs:caching-system ref="caching-system-id"/>
    <wlevs:cache-listener ref="cache-listener-id" />
```

```
</wlevs:cache>
...
<bean id="cache-listener-id" class="wlevs.example.MyCacheListener"/>
```

In the example, the `cache-listener-id` Spring bean listens to events coming from the cache; the class that implements this component, `wlevs.example.MyCacheListener`, must implement the `com.bea.jcache.CacheListener` interface. You must program the `wlevs.example.MyCacheListener` class yourself.

wlevs:cache-loader

Specifies the Spring bean that implements an object that loads data into a cache.

This tag is always a child of wlevs:cache.

Attributes

The following table lists the attributes of the `wlevs:cache-loader` application assembly tag.

Table 2-4 Attributes of the wlevs:cache-loader Application Assembly Tag

Attribute	Description	Data Type	Required?
ref	Specifies the Spring bean that implements the class that loads data into the cache. Set this attribute to the value of the `id` attribute of the Spring bean. The Spring bean must implement the `com.bea.cache.jcache.CacheLoader` interface.	String	Yes.

Example

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

```
<wlevs:cache id="cache-id" name="alternative-cache-name">
    <wlevs:caching-system ref="caching-system-id"/>
    <wlevs:cache-loader ref="cache-loader-id" />
</wlevs:cache>
```

```
  ...
<bean id="cache-loader-id" class="wlevs.example.MyCacheLoader"/>
```

In the example, the `cache-loader-id` Spring bean, implemented with the `wlevs.example.MyCacheLoader` class that in turn implements the `com.bea.cache.jcache.CacheLoader` interface, is a bean that loads data into a cache. The cache specifies this loader by pointing to it with the `ref` attribute of the `<wlevs:cache-loader>` child element.

wlevs:cache-source

Specifies a cache that supplies data to this processor component. The processor component in turn is associated with an EPL query that directly references the cache.

Use the `value-type` attribute of the wlevs:cache tag to declare the event type of the data supplied by the cache.

This tag is a child of only wlevs:processor tag.

Attributes

The following table lists the attributes of the `wlevs:cache-source` application assembly tag.

Table 2-5 Attributes of the wlevs:cache-source Application Assembly Tag

Attribute	Description	Data Type	Required?
ref	Specifies the cache that is a source of data for the processor component. Set this attribute to the value of the `id` attribute of the cache.	String	Yes.

Example

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

```
<wlevs:caching-system id="caching-system-id"/>
 ...
 <wlevs:cache id="cache-id"
              name="alternative-cache-name"
```

```
                value-type="Company">
  <wlevs:caching-system ref="caching-system-id"/>
</wlevs:cache>

<wlevs:stream id="stream-id"/>

<wlevs:processor id="processor-id">
  <wlevs:cache-source ref="cache-id">
  <wlevs:source ref="stream-id">
</wlevs:processor>
```

In the example, the processor will have data pushed to it from the `stream-id` stream as usual; however, the EPL queries that execute in the processor can also pull data from the `cache-id` cache. When the query processor matches an event type in the FROM clause to an event type supplied by a cache, such as `Company`, the processor pulls instances of that event type from the cache.

wlevs:cache-store

Specifies the Spring bean that implements a custom store that is responsible for writing data from the cache to a backing store, such as a table in a database.

This tag is always a child of wlevs:cache.

Attributes

The following table lists the attributes of the `wlevs:cache-store` application assembly tag.

Table 2-6 Attributes of the wlevs:cache-store Application Assembly Tag

Attribute	Description	Data Type	Required?
ref	Specifies the Spring bean that implements the custom store. Set this attribute to the value of the `id` attribute of the Spring bean. The Spring bean must implement the `com.bea.cache.jcache.CacheStore` interface.	String	Yes.

Example

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

```
    <wlevs:cache id="cache-id" name="alternative-cache-name">
        <wlevs:caching-system ref="caching-system-id"/>
        <wlevs:cache-store ref="cache-store-id" />
    </wlevs:cache>
    ...
   <bean id="cache-store-id" class="wlevs.example.MyCacheStore"/>
```

In the example, the `cache-store-id` Spring bean, implemented with the `wlevs.example.MyCacheStore` class that in turn implements the `com.bea.cache.jcache.CacheStore` interface, is a bean for the custom store, such as a database. The cache specifies this store by pointing to it with the `ref` attribute of the `<wlevs:cache-store>` child element.

wlevs:caching-system

Specifies the caching system used by the application.

Child Tags

The `wlevs:caching-system` application assembly tag supports the following child tag:

- wlevs:instance-property
- wlevs:property

Attributes

The following table lists the attributes of the `wlevs:caching-system` application assembly tag.

Table 2-7 Attributes of the wlevs:caching-system Application Assembly Tag

Attribute	Description	Data Type	Required?
id	Specifies the unique identifier for this caching system. This identifier must correspond to the `<name>` element in the XML configuration file for this caching system	String	Yes.
advertise	Advertises this service in the OSGi registry. Valid values are `true` and `false`. Default value is `false`.	Boolean	No.
provider	Specifies the provider of the caching system if you are using a third-party implementation, such as Oracle Coherence. Typically this attribute corresponds to the `provider-name` attribute of a `<factory>` Spring tag that specifies the factory class that creates instances of the third-party caching system. If you do not specify the `provider` or `class` attribute, then the default value is Oracle CEP's own caching implementation for local single-JVM caches; this implementation uses an in-memory store.	String	No.
class	Specifies the Java class that implements this caching system; use this attribute to specify a third-party implementation rather than Oracle CEP's own implementation. If you specify this attribute, it is assumed that the third-party implementation code resides inside the Oracle CEP application bundle itself. The class file to which this attribute points must implement the `com.bea.wlevs.cache.api.CachingSystem` interface. If you do not specify the `provider` or `class` attribute, then the default value is Oracle CEP's own caching implementation for local single-JVM caches; this implementation uses an in-memory store.	String	No

Example

The following example shows the simplest use of the `wlevs:caching-system` tag in the EPN assembly file:

```
<wlevs:caching-system id="caching-system-id"/>
```

The following example shows how to specify a third-party implementation that uses a factory as a provider:

```
<wlevs:caching-system id ="caching-system-id"
                      provider="caching-provider"/>

<factory id="factory-id" provider-name="caching-provider">
   <class>the.factory.class.name</class>
</factory>
```

In the example, `the.factory.class.name` is a factory for creating some third-party caching system; the provider attribute of wlevs:caching-system in turn references it as the caching system implementation for the application.

wlevs:event-bean

Use this tag to declare to the Spring application context that an event bean is part of your event processing network (EPN). Event beans are managed by the Oracle CEP container, analogous to Spring beans that are managed by the Spring framework. In many ways, event beans and Spring beans are similar so it is up to a developer which one to use in their EPN. Use a Spring bean for legacy integration to Spring. Use an event bean if you want to take full advantage of the additional capabilities of Oracle CEP.

For example, you can monitor an event bean using the Oracle CEP monitoring framework, make use of the Configuration framework metadata annotations, and record and playback events that pass through the event bean. An event-bean can also participate in the Oracle CEP bean lifecycle by specifying methods in its EPN assembly file declaration, rather than by implementing Oracle CEP API interfaces.

Child Tags

The `wlevs:event-bean` application assembly tag supports the following child tags:

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

Attributes

The following table lists the attributes of the `wlevs:event-bean` application assembly tag.

Table 2-8 Attributes of the wlevs:event-bean Application Assembly Tag

Attribute	Description	Data Type	Required?
id	Unique identifier for this component. This identifier must correspond to the `<name>` element in the XML configuration file for this event-bean, if one exists.	String	Yes.
advertise	Advertises this service in the OSGi registry. Valid values are `true` and `false`. 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.
class	Specifies the Java class that implements this event bean. The bean is not required to implement any Oracle CEP interfaces. You must specify either the `provider` or `class` attribute, but not both, otherwise an exception is raised.		
provider	Specifies the service provider. In this case, an EDE factory registered with this specific provider name must exist in the application. You must specify either the `provider` or `class` attribute, but not both, otherwise an exception is raised.	String	No.
onevent-method	Specifies the method of the event bean implementation that corresponds to the lifecycle `onEvent` method. Oracle CEP invokes this method when the event bean receives an event. By using this lifecycle attribute, the event bean implementation does not have to explicitly implement an Oracle CEP interface.	String	No

Table 2-8 Attributes of the wlevs:event-bean Application Assembly Tag

Attribute	Description	Data Type	Required?
init-method	Specifies the method of the event bean implementation that corresponds to the lifecycle `init` method. Oracle CEP invokes this method after it has set all the supplied instance properties. This method allows the bean instance to perform initialization only possible when all bean properties have been set and to throw an exception in the event of misconfiguration. By using this lifecycle attribute, the event bean implementation does not have to explicitly implement an Oracle CEP interface.	String	No
activate-method	Specifies the method of the event bean implementation that corresponds to the lifecycle `activate` method. Oracle CEP invokes this method after the dynamic configuration of the bean has completed. This method allows the bean instance to perform initialization only possible when all dynamic bean properties have been set and the EPN has been wired. By using this lifecycle attribute, the event bean implementation does not have to explicitly implement an Oracle CEP interface.	String	No
suspend-method	Specifies the method of the event bean implementation that corresponds to the lifecycle `suspend` method. Oracle CEP invokes this method when the application is suspended. By using this lifecycle attribute, the event bean implementation does not have to explicitly implement an Oracle CEP interface.	String	No
destroy-method	Specifies the method of the event bean implementation that corresponds to the lifecycle `destroy` method. Oracle CEP invokes this method when the application is stopped. By using this lifecycle attribute, the event bean implementation does not have to explicitly implement an Oracle CEP interface.	String	No

Example

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

```
<wlevs:event-bean id="myBean" class="com.customer.SomeEventBean" >
  <wlevs:listener ref="myProcessor" />
</wlevs:event-bean>
```

In the example, the event bean called `myBean` is implemented with the class `com.customer.SomeEventBean`. The component called `myProcessor` receives events from the `myBean` event bean.

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` application assembly 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 2-21 for additional details.

wlevs:event-type

Specifies the definition of an event type used in the Oracle CEP 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 Oracle CEP 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.

Oracle 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` application assembly tag supports the following child tags:

- wlevs:metadata
- wlevs:property

Attributes

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

Table 2-9 Attributes of the wlevs:event-type Application Assembly Tag

Attribute	Description	Data Type	Required?
id	Specifies the unique identifier for this event type. If you do not specify this attribute, Oracle CEP 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:factory

Use this tag to register a factory class as a service. Use of this tag decreases the dependency of your application on Spring-OSGi interfaces.

The Java source of this factory must implement the `com.bea.wlevs.ede.api.Factory` interface.

The factory tag does not allow you to specify service properties. If you need to specify service properties, then you must use the Spring- OSGi `<osgi:service>` tag instead.

This tag does not have any child tags.

Attributes

The following table lists the attributes of the `wlevs:factory` application assembly tag.

Table 2-10 Attributes of the wlevs:factory Application Assembly Tag

Attribute	Description	Data Type	Required?
class	Specifies the Java class that implements the factory. This class must implement the `com.bea.wlevs.ede.api.Factory` interface.	String	Yes.
provider-name	Specifies the name of this provider. Reference this name later in the component that uses this factory.	String	Yes.

Example

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

```
<wlevs:factory provider-name="myEventSourceFactory"
                class="com.customer.MyEventSourceFactory" />
```

In the example, the factory implemented by the `com.customer.MyEventSourceFactory` goes by the provider name of `myEventSourceFactory`.

wlevs:function

Use this tag to specify a bean that contains user-defined functions for a processor.

This tag always has a standard Spring `<bean>` tag as a child that specifies the Spring bean for the user-defined function.

Attributes

The following table lists the attributes of the `wlevs:function` application assembly tag.

Table 2-11 Attributes of the wlevs:function Application Assembly Tag

Attribute	Description	Data Type	Required?
epl-name	An alternate name to use when referencing this function bean in an EPL query. The default value is the Spring bean name.	String	No.

Example

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

```
<wlevs:processor id="testProcessor">
  <wlevs:listener ref="providerCache"/>
  <wlevs:listener ref="outputCache"/>
  <wlevs:cache-source ref="testCache"/>
  <wlevs:function epl-name="testfunction">
    <bean class="com.bea.wlevs.example.cache.function.TestFunction"/>
  </wlevs:function>
</wlevs:processor>
```

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, wlevs:stream, or wlevs:caching-system.

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 application assembly tag.

Table 2-12 Attributes of the wlevs:instance-property Application Assembly 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, wlevs:stream, wlevs:event-bean or wlevs:cache.

Attributes

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

Table 2-13 Attributes of the wlevs:listener Application Assembly 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, Oracle CEP 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` application assembly tag.

Table 2-14 Attributes of the wlevs:metadata Application Assembly 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
- wlevs:source
- wlevs:function

Attributes

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

Table 2-15 Attributes of the wlevs:processor Application Assembly Tag

Attribute	Description	Data Type	Required?
id	Unique identifier for this component. This identifier must correspond to the `<name>` element in the XML configuration file for this processor; this is how Oracle CEP 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.
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.
queryURL	Specifies a URL that points to an EPL rules definition file for this processor.	String.	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 Oracle CEP 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, wlevs:stream, or wlevs:caching-system.

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` application assembly tag.

Table 2-16 Attributes of the wlevs:property Application Assembly Tag

Attribute	Description	Data Type	Required?
name	Specifies the name of the property, following JavaBean naming conventions.	String	Yes.

Table 2-16 Attributes of the wlevs:property Application Assembly Tag

Attribute	Description	Data Type	Required?
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 component, or in other words, the component which the events are coming *from*. Specifying an event source is equivalent to specifying this component 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 a child of wlevs:stream or wlevs:processor.

Attributes

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

Table 2-17 Attributes of the wlevs:source Application Assembly 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` application assembly 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` application assembly tag.

Table 2-18 Attributes of the wlevs:stream Application Assembly Tag

Attribute	Description	Data Type	Required?
id	Unique identifier for this component. This identifier must correspond to the `<name>` element in the XML configuration file for this stream, if one exists.	String	Yes.
advertise	Advertises this service in the OSGi registry. Valid values are `true` and `false`. 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.
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.

Table 2-18 Attributes of the wlevs:stream Application Assembly Tag

Attribute	Description	Data Type	Required?
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

The `Deployer` is a Java-based deployment utility that provides administrators and developers command-line based operations for deploying Oracle CEP applications. In the context of Oracle CEP 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 Oracle CEP 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 an Oracle CEP Application: Main Steps for detailed instructions on creating this deployment bundle.

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

Oracle CEP uses the `deployments.xml` file to internally maintain its list of deployed application OSGi bundles. This file is located in in the *`DOMAIN_DIR/servername`* directory, where *`DOMAIN_DIR`* refers to the main domain directory correspoding to the server instance to which you are deploying your application and *`servername`* refers to the server instance itself. See "Deployment XSD Schema" on page 5-3 for information about this file.

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

Required Environment for the Deployer Utility

To set up your environment to use the `Deployer` utility:

1. Install and configure the Oracle CEP software, as described in Installing Oracle CEP.
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 `wlevsdeploy.jar JAR` file, located in the *`ORACLE_CEP_HOME`*`/ocep_10.3/bin` directory where, *`ORACLE_CEP_HOME`* refers to the main Oracle CEP installation directory, such as `/oracle_cep`.

Running the Deployer Utility Remotely

Sometimes it is useful to run the `Deployer` utility on a computer different from the computer on which Oracle CEP is installed and running. To run the utility remotely, follow these steps:

1. Copy the following JAR files from the computer on which Oracle CEP 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:

   – *ORACLE_CEP_HOME*/ocep_10.3/bin/wlevsdeploy.jar

   where *ORACLE_CEP_HOME* refers to the main directory into which you installed Oracle CEP.

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*/`wlevsdeploy.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 `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 `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 -jar wlevsdeploy.jar
      [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 3-8 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 Oracle CEP 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 Oracle CEP is running.
	• *`port`* refers to the port number to which Oracle CEP listens; its value is `9002` by default. This port is specified in the `config.xml` file that describes your Oracle CEP 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 Oracle CEP 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 Oracle CEP 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 Oracle CEP 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 Oracle CEP instance. The *`bundle`* parameter refers to a filename that is local to the computer from which you execute the `Deployer` utility. 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, Oracle CEP: • 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. • Starts the application. The incoming adapters immediately start receiving data.
`-update` *`bundle`*	Updates the existing OSGi bundle with new application code. The *`bundle`* parameter refers to a filename that is local to the computer from which you execute the `Deployer` utility. 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, Oracle CEP: • 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.

Table 3-3 Deployment Commands

Command	Description
`-uninstall` *`name`*	Removes the existing bundle from the specified Oracle CEP 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, Oracle CEP: • Removes the specified OSGi bundle from the domain directory. • Removes the bundles from the internal deployment registry .
`-suspend` *`name`*	Suspends a currently running OSGI bundle which was previously installed to the specified Oracle CEP 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`
`-resume` *`name`*	Resumes a previously suspended OSGI bundle on the specified Oracle CEP instance; the configured adapters once again start immediately receiving incoming data. 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`

`-group` *`groupname`*	Specifies that the deploy command (install, uninstall, update, suspend, or resume) applies to a target group, or more specifically, to the set of running servers within that group.
	To specify the domain group, use the keyword `all`, such as:
	`-group all`
	To specify a custom group, simply specify the name of the group:
	`-group my_group`

Table 3-3 Deployment Commands

Command	Description
`-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, Oracle CEP 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 -jar wlevsdeploy.jar
        -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 resume this application after it has been suspended:

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

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
```

The following example shows how to install an application called `strategies_1.0.jar` to the `strategygroup`; this example also shows how to use the `-jar` command of the `java` utility:

```
prompt> java -jar wlevsdeploy.jar
        -url http://ariel:9002/wlevsdeployer -install strategies_1.0.jar
         -group strategygroup
```

CHAPTER 4

Metadata Annotations

This section contains information on the following subjects:

Overview of Oracle Complex Event Processing Metadata Annotations

The Oracle Complex Event Processing (or *Oracle CEP* for short) 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 Oracle CEP uses to send configuration information to the adapter.

Oracle CEP 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, Oracle CEP 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>
```

Oracle CEP 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 Oracle CEP uses to send configuration information to the adapter.

Oracle CEP 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, Oracle CEP 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; Oracle CEP 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 Oracle CEP uses to send configuration information to the adapter.

Oracle CEP 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, Oracle CEP 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; Oracle CEP 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 Oracle CEP 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 Oracle Complex Event Processing (or *Oracle CEP* for short) components, such as the complex event processors and adapters.

The `wlevs_application_config.xsd` schema imports both the `wlevs_base_config.xsd` and `wlevs_eventstore_config.xsd` schemas.

- wlevs_application_config.xsd
- wlevs_eventstore_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"?>
<n1:config xmlns:n1="http://www.bea.com/ns/wlevs/config/application"
           xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">

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

  <stream>
      <name>helloworldOutstream</name>
      <max-size>10000</max-size>
      <max-threads>2</max-threads>
  </stream>

</n1:config>
```

EPN Assembly XSD Schema

You use the EPN assembly file to declare the components that make up your Oracle CEP 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 Oracle CEP-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">

    <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>

    <wlevs:adapter id="helloworldAdapter"

class="com.bea.wlevs.adapter.example.helloworld.HelloWorldAdapter" >
        <wlevs:instance-property name="message"
                                 value="HelloWorld - the currenttime is:"/>
    </wlevs:adapter>

    <wlevs:processor id="helloworldProcessor" />

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

    <wlevs:stream id="helloworldOutstream" advertise="true">
        <wlevs:listener>
            <bean class="com.bea.wlevs.example.helloworld.HelloWorldBean"/>
        </wlevs:listener>
        <wlevs:source ref="helloworldProcessor"/>
    </wlevs:stream>

</beans>
```

Deployment XSD Schema

The deployment file for an Oracle CEP instance is called `deployments.xml` and is located in the *`DOMAIN_DIR/servername`* directory, where *`DOMAIN_DIR`* refers to the main domain directory and *`servername`* refers to the name of the server instance. 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">

  <bean
class="org.springframework.beans.factory.config.PropertyPlaceholderConfigurer"
>
      <property name="systemPropertiesModeName"
value="SYSTEM_PROPERTIES_MODE_OVERRIDE"/>
  </bean>

  <wlevs:deployment id="fx" state="start"
location="file:${wlevs.domain.home}/applications/fx/com.bea.wlevs.example.fx_3
.0.0.0.jar"/>

</beans>
```

Server Configuration XSD Schema

The Oracle CEP server configuration file, `config.xml`, is located in the *`DOMAIN_DIR`*`/`*`servername`*`/config` directory, where *`DOMAIN_DIR`* refers to the main domain directory and *`servername`* refers to a particular server instance. To change the configuration of an Oracle CEP instance, you can update this file manually and add or remove server configuration elements.

See wlevs_server_config.xsd for the full XSD Schema.

Example of a Server Configuration XML File

The following sample `config.xml`, from the *`ORACLE_CEP_HOME`*`/user_projects/domains/wlevs30_domain/defaultserver` template domain, shows how to configure some of these services:

```
<?xml version="1.0" encoding="UTF-8"?>
<n1:config xsi:schemaLocation="http://www.bea.com/ns/wlevs/config/server
```

```
wlevs_server_config.xsd"
 xmlns:n1="http://www.bea.com/ns/wlevs/config/server"
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">

  <netio>
    <name>NetIO</name>
    <port>9002</port>
  </netio>

  <netio>
    <name>sslNetIo</name>
    <ssl-config-bean-name>sslConfig</ssl-config-bean-name>
    <port>9003</port>
  </netio>

  <work-manager>
    <name>JettyWorkManager</name>
    <min-threads-constraint>5</min-threads-constraint>
    <max-threads-constraint>10</max-threads-constraint>
  </work-manager>

  <jetty>
    <name>JettyServer</name>
    <network-io-name>NetIO</network-io-name>
    <work-manager-name>JettyWorkManager</work-manager-name>
    <secure-network-io-name>sslNetIo</secure-network-io-name>
  </jetty>

  <rmi>
    <name>RMI</name>
    <http-service-name>JettyServer</http-service-name>
  </rmi>

  <jndi-context>
   <name>JNDI</name>
  </jndi-context>

  <exported-jndi-context>
    <name>exportedJndi</name>
    <rmi-service-name>RMI</rmi-service-name>
  </exported-jndi-context>
```

```
  <jmx>
    <rmi-service-name>RMI</rmi-service-name>
    <rmi-jrmp-port>9999</rmi-jrmp-port>
    <jndi-service-name>JNDI</jndi-service-name>
    <rmi-registry-port>9004</rmi-registry-port>
  </jmx>
  <ssl>
    <name>sslConfig</name>
    <key-store>./ssl/dsidentity.jks</key-store>
    <key-store-pass>
        <password>changeit</password>
    </key-store-pass>
    <key-store-alias>ds</key-store-alias>
    <key-manager-algorithm>SunX509</key-manager-algorithm>
    <ssl-protocol>TLS</ssl-protocol>
    <enforce-fips>false</enforce-fips>
    <need-client-auth>false</need-client-auth>
  </ssl>
  <http-pubsub>
    <name>pubsub</name>
    <path>/pubsub</path>
    <pub-sub-bean>
      <server-config>
        <name>pubsubbean</name>
        <supported-transport>
          <types>
            <element>long-polling</element>
          </types>
        </supported-transport>

<publish-without-connect-allowed>true</publish-without-connect-allowed>
      </server-config>
    <channels>
        <element>
          <channel-pattern>/evsmonitor</channel-pattern>
        </element>
        <element>
```

```
          <channel-pattern>/evsalert</channel-pattern>
        </element>
        <element>
          <channel-pattern>/evsdomainchange</channel-pattern>
        </element>
      </channels>
    </pub-sub-bean>
  </http-pubsub>

</n1:config>
```