Oracle® CEP

IDE Developer's Guide for Eclipse

Release 11*g*R1 (11.1.1)

E14301-01

May 2009

ORACLE®

Oracle CEP IDE Developer's Guide for Eclipse Release 11gR1 (11.1.1)

E14301-01

Contents

1 Overview of Creating Oracle CEP Applications

2 Overview of the Oracle CEP IDE for Eclipse

3 Oracle CEP IDE for Eclipse Projects

4 Oracle CEP IDE for Eclipse and Oracle CEP Servers

5   Oracle CEP IDE for Eclipse and the Event Processing Network

6   Configuring JMS Adapters

7 Configuring HTTP Publish-Subscribe Server Adapters

8 Configuring Custom Adapters, Event Beans, and Spring Beans

9 Configuring Channels

10 Configuring Oracle CQL Processors

11 Configuring EPL Processors

12 Configuring Caching

13   Configuring Event Record and Playback

14 Assembling and Deploying Oracle CEP Applications

15 Testing Applications With the Load Generator and csvgen Adapter

A Additional Information about Spring and OSGi

B Oracle CEP Schemas

C Schema Reference: Component Configuration wlevs_application-config.xsd

D   Schema Reference: EPN Assembly spring-wlevs-v11_0_0_0.xsd

E   Schema Reference: Deployment deployment.xsd

F   Schema Reference: Server Configuration wlevs_server_config.xsd

G Oracle CEP Metadata Annotation Reference

H Oracle CEP IDE for Eclipse Tutorial

Index

List of Examples

List of Figures

List of Tables

Preface

This document describes how to create, deploy, and debug Oracle CEP applications.

Oracle CEP (formally known as the WebLogic Event Server) is a Java server for the development of high-performance event driven applications. It is a lightweight Java application container based on Equinox OSGi, with shared services, including the Oracle CEP Service Engine, which provides a rich, declarative environment based on Oracle Continuous Query Language (Oracle CQL) - a query language based on SQL with added constructs that support streaming data - to improve the efficiency and effectiveness of managing business operations. Oracle CEP supports ultra-high throughput and microsecond latency using JRockit Real Time and provides Oracle CEP Visualizer and Oracle CEP IDE for Eclipse developer tooling for a complete real time end-to-end Java Event-Driven Architecture (EDA) development platform.

Audience

This document is intended for programmers creating Oracle CEP applications.

Documentation Accessibility

Our goal is to make Oracle products, services, and supporting documentation accessible to all users, including users that are disabled. To that end, our documentation includes features that make information available to users of assistive technology. This documentation is available in HTML format, and contains markup to facilitate access by the disabled community. Accessibility standards will continue to evolve over time, and Oracle is actively engaged with other market-leading technology vendors to address technical obstacles so that our documentation can be accessible to all of our customers. For more information, visit the Oracle Accessibility Program Web site at http://www.oracle.com/accessibility/.

Accessibility of Code Examples in Documentation

Screen readers may not always correctly read the code examples in this document. The conventions for writing code require that closing braces should appear on an otherwise empty line; however, some screen readers may not always read a line of text that consists solely of a bracket or brace.

Accessibility of Links to External Web Sites in Documentation

This documentation may contain links to Web sites of other companies or organizations that Oracle does not own or control. Oracle neither evaluates nor makes any representations regarding the accessibility of these Web sites.

Deaf/Hard of Hearing Access to Oracle Support Services

To reach Oracle Support Services, use a telecommunications relay service (TRS) to call Oracle Support at 1.800.223.1711. An Oracle Support Services engineer will handle technical issues and provide customer support according to the Oracle service request process. Information about TRS is available at `http://www.fcc.gov/cgb/consumerfacts/trs.html`, and a list of phone numbers is available at `http://www.fcc.gov/cgb/dro/trsphonebk.html`.

Related Documents

For more information, see the following:

- *Oracle CEP Getting Started*

- *Oracle CEP Administrator's Guide*

- *Oracle CEP Visualizer User's Guide*

- *Oracle CEP Java API Reference*

- *Oracle CEP CQL Language Reference*

- *Oracle CEP EPL Language Reference*

- *Oracle Database SQL Language Reference*

- SQL99 Specifications (ISO/IEC 9075-1:1999, ISO/IEC 9075-2:1999, ISO/IEC 9075-3:1999, and ISO/IEC 9075-4:1999)

- Oracle Event Driven Architecture Suite sample code: `http://www.oracle.com/technology/sample_code/products/event-driven-architecture`

Conventions

The following text conventions are used in this document:

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

Part I

Introduction

Part I contains the following chapters:

- Section 1, "Overview of Creating Oracle CEP Applications"

1

Overview of Creating Oracle CEP Applications

This section contains information on the following subjects:

- Section 1.1, "Overview of the Oracle CEP Programming Model"
- Section 1.2, "Oracle CEP IDE for Eclipse"
- Section 1.3, "Creating an Oracle CEP Application"
- Section 1.4, "Creating the EPN Assembly File"
- Section 1.5, "Creating Oracle CEP Event Types"
- Section 1.6, "Configuring Oracle CEP Resource Access"
- Section 1.7, "Next Steps"

1.1 Overview of the Oracle CEP Programming Model

Because Oracle CEP applications are low latency high-performance driven applications, they run on a lightweight container and are developed using a POJO-based programming model. In POJO (Plain Old Java Object) programming, business logic is implemented in the form of POJOs, and then injected with the services they need. This is popularly called *dependency injection*. The injected services can range from those provided by Oracle CEP services, such as configuration management, to those provided by another Oracle product such as Oracle Kodo, to those provided by a third party.

Oracle CEP defines a set of core services or components used together to assemble event-driven applications; the typical services are adapters, streams, and processors. You can also create your own business logic POJOs and Spring beans that are part of the application, as well as specialized event beans that are just like Spring beans but with full access to the Oracle CEP framework, such as monitoring and record/playback of events. In addition to these, Oracle CEP includes other infrastructure services, such as caching, clustering, configuration, monitoring, logging, and so on.

All services are deployed on the underlying Oracle microServices Architecture (mSA) technology. Oracle mSA is based upon the OSGi Service Platform defined by the OSGi Alliance (see `http://www.osgi.org/` for more information.)

The following sections provide additional information about the Oracle CEP programming model and creating applications:

- Section 1.1.1, "Components of the Oracle CEP Event Processing Network"

- Section 1.1.2, "Event Types"

- Section 1.1.3, "Event Type Data Types"

- Section 1.1.4, "Stream Sources and Stream Sinks and Relation Sources and Relation Sinks"

- Section 1.1.5, "Component Configuration Files"

- Section 1.1.6, "Accessing Component and Server Configuration Using the ConfigurationPropertyPlaceholderConfigurer Class"

- Section 1.1.7, "EPN Assembly File"

- Section 1.1.8, "How Components Fit Together"

- Section 1.1.9, "Oracle CEP Application Lifecycle"

- Section 1.1.10, "Oracle CEP APIs"

1.1.1 Components of the Oracle CEP Event Processing Network

The Oracle CEP Event Processing Network (EPN) represents the interconnections between the various Oracle CEP components of an Oracle CEP application.

Oracle CEP applications and their event processing networks (EPNs) are made up of the following basic components:

- Adapter: Components that provide an interface to incoming data feeds and convert the data into event types that the Oracle CEP application understands.

  Adapters can be both incoming (receive data) and outgoing (send data). Oracle CEP includes some built-in adapters, such as HTTP publish-subscribe adapters.

  For more information, see:

  - Chapter 6, "Configuring JMS Adapters"

  - Chapter 7, "Configuring HTTP Publish-Subscribe Server Adapters"

  - Chapter 8, "Configuring Custom Adapters, Event Beans, and Spring Beans"

- Channel: Components that represent the physical conduit through which events flow.

  Channels connect components that send events with components that receive events. For more information, see Chapter 9, "Configuring Channels".

- Processor: Components that execute user-defined event processing rules against the events offered by channels.

  The user-defined rules are written using either of the following:

  - Oracle Continuous Query Language (Oracle CQL)

    For more information, see:

    * Chapter 10, "Configuring Oracle CQL Processors"

    * *Oracle CEP CQL Language Reference*

  - Event Processing Language (EPL)

    For more information, see:

    * Chapter 11, "Configuring EPL Processors"

    * *Oracle CEP EPL Language Reference*

> **Note:** Oracle CQL replaces Event Processing Language (EPL) in Oracle CEP Release 11*g*R1 (11.1.1). Oracle CEP supports EPL for backwards compatibility.

- Event bean: User-coded Plain Old Java Object (POJO) bean that is managed by Oracle CEP.

  Event beans are analogous to Spring beans, which are managed by the Spring framework, however they support event-based features, such as the specification of a suspend and resume methods. For more information, see Chapter 8, "Configuring Custom Adapters, Event Beans, and Spring Beans".

- Spring bean: User-coded Plain Old Java Object (POJO) managed by the Spring framework.

  These components are not managed by Oracle CEP; for example, you cannot monitor these components, record and playback of events, and so on. If you require this additional functionality for your POJO, consider creating an Oracle CEP event bean instead. For more information, see Section 8.1.3, "Custom Spring Beans".

- Cache: Temporary storage area for events, created exclusively to improve the overall performance of your application.

  For more information, see:

  - Section 10.4, "Configuring an Oracle CQL Processor Cache Source"

  - Section 11.3, "Configuring an EPL Processor Cache Source"

  - Chapter 12, "Configuring Caching".

- Table: A node that connects a relational database table to the EPN as an event data source.

  For more information, see Section 10.3, "Configuring an Oracle CQL Processor Table Source".

For more information, see:

- Section 5.4.1, "Creating Nodes"

- Section 1.1.1.1, "Nested Stages"

- Section 1.1.1.2, "Foreign Stages"

1.1.1.1 Nested Stages

When you define a child stage within a parent stage, the child stage is said to be nested. Only the parent stage can specify the child stage as a listener.

Example 1–1 shows the EPN assembly source in which `HelloWorldBean` is nested within the `helloworldOutputChannel`. Only the parent `helloworldOutputChannel` may specify the nested bean as a listener.

Example 1–1   EPN Assembly File With Nested Bean

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

<wlevs:channel id="helloworldInputChannel" event-type="HelloWorldEvent" >
```

```
                        <wlevs:listener ref="helloworldProcessor"/>
                        <wlevs:source ref="helloworldAdapter"/>
                    </wlevs:channel>

                    <wlevs:processor id="helloworldProcessor" />

                    <wlevs:channel id="helloworldOutputChannel" event-type="HelloWorldEvent" advertise="true">
                        <wlevs:listener>
                            <bean class="com.bea.wlevs.example.helloworld.HelloWorldBean"/>
                        </wlevs:listener>
                        <wlevs:source ref="helloworldProcessor"/>
                    </wlevs:channel>
```

Alternatively, you can define this EPN so that all nodes are nested as Example 1–2 shows. The `helloworldAdapter`, the outermost parent stage, is the only stage accessible to other stages in the EPN.

Example 1–2   EPN Assembly File With all Nodes Nested

```
<wlevs:adapter id="helloworldAdapter" class="com.bea.wlevs.adapter.example.helloworld.HelloWorldAdapter" >
    <wlevs:instance-property name="message" value="HelloWorld - the current time is:"/>
    <wlevs:listener>
        <wlevs:channel id="helloworldInputChannel" event-type="HelloWorldEvent" >
            <wlevs:listener>
                <wlevs:processor id="helloworldProcessor">
                    <wlevs:listener>
                        <wlevs:channel id="helloworldOutputChannel" event-type="HelloWorldEvent">
                            <wlevs:listener>
                                <bean class="com.bea.wlevs.example.helloworld.HelloWorldBean"/>
                            </wlevs:listener>
                        </wlevs:channel>
                    </wlevs:listener>
                </wlevs:processor>
            </wlevs:listener>
        </wlevs:channel>
    </wlevs:listener>
</wlevs:adapter>
```

For more information, see Section 5.2.6, "Nested Stages".

1.1.1.2 Foreign Stages

You can refer to a stage simply by its `id` attribute when you define both the source and target stage in the same application.

To refer to a stage you define in a different application, you use the following syntax:

FOREIGN-APPLICATION-NAME:*FOREIGN-STAGE-ID*

Where *FOREIGN-APPLICATION-NAME* is the name of the application in which you defined the foreign stage and *FOREIGN-STAGE-ID* is the `id` attribute of the foreign stage.

Example 1–3 shows how the reference in `application1` to the foreign stage `HelloWorldBeanSource` that you define in application `application2`.

Example 1–3   Application 1 Referencing Foreign Stage in Application 2

```
<wlevs:stream id="helloworldInstream" >
    <wlevs:listener ref="helloworldProcessor"/>
    <wlevs:source ref="application2:HelloWorldBeanSource"/>
</wlevs:stream>
```

Example 1–4   Foreign Stage in Application 2

```
<wlevs:event-bean id="HelloWorldBeanSource"
class="com.bea.wlevs.example.helloworld.HelloWorldBeanSource"
advertise="true"/>
```

1.1.2 Event Types

Event types define the properties of the events that are handled by Oracle CEP applications. Adapters receive incoming events from different event sources, such as JMS, or financial market data feeds. You must define these events by an event type before a processor is able to handle them. You then use these event types in the adapter and POJO Java code, as well as in the Oracle CQL and EPL rules you associate with the processors.

Events are JavaBean or Java class instances in which each property represents a data item from the event source.

Oracle recommends that you create your own JavaBean class that represents the event type and register the class in the EPN assembly file. By creating your own JavaBean, you can reuse it and you have complete control over what the event looks like. If you cannot represent your event as a JavaBean, you represent it as a Java class and provide a factory to build instances of the event.

Alternatively, you can specify the properties of the event type declaratively in the EPN assembly file using `wlevs:property` tags, in which case Oracle CEP will use a `java.util.Map` or tuple. Tuple is the default and is similar to the Oracle CQL tuple. It essentially provides the user with an optimized implementation, however, the user must always set and get its value using the Event Type Repository APIs. This approach is best used for quick prototyping

You can also create an event type programmatically using the `EventTypeRepository` class.

When creating event types, observe the data type restrictions that Section 1.1.3, "Event Type Data Types" describes.

For more information, see Section 1.5, "Creating Oracle CEP Event Types".

1.1.3 Event Type Data Types

When creating event types, observe the data type restrictions for the following event types:

- Section 1.1.3.1, "Event Types Specified as Java Bean, Java Class, or java.util.Map"

- Section 1.1.3.2, "Event Types Specified as a Tuple"

- Section 1.1.3.3, "Event Types for use With a Table Source"

- Section 1.1.3.4, "Event Types for use With the csvgen Adapter"

For more information, see:

- Section 1.1.2, "Event Types"

- "Datatypes" in the *Oracle CEP CQL Language Reference*

- "Overview of the EPL Language" in the *Oracle CEP EPL Language Reference*

1.1.3.1 Event Types Specified as Java Bean, Java Class, or java.util.Map

When you define an event type as a Java Bean, Java class, `java.util.Map`, you may use any Java type for its properties.

For more information, see

- Section 1.5.1, "How to Create an Oracle CEP Event Type as a JavaBean"
- Section 1.5.2, "How to Create an Oracle CEP Event Type as a Java Class"
- Section 1.5.3, "How to Create an Oracle CEP Event Type as a java.util.Map"

1.1.3.2 Event Types Specified as a Tuple

When you specify the properties of the event type declaratively in the EPN assembly file as a tuple, you may only use the types specified by the `com.bea.wlevs.ede.api.Type` class. Table 1–1 describes the usage for each of the types this class defines.

Table 1–1   com.bea.wlevs.ede.api.Type Usage

Type	Usage
bigint	Numeric values in the range that `java.math.BigInteger` specifies.
boolean	Boolean values as `java.lang.Boolean` specifies.
byte	Byte values as `java.lang.Byte` specifies.
char	Single or multiple character values. Use for both `char` and `java.lang.String` values.
double	Numeric values in the range that `java.lang.Double` specifies.
float	Numeric values in the range that `java.lang.Float` specifies.
int	Numeric values in the range that `java.lang.Integer` specifies.
interval	Interval values as the Oracle CQL `INTERVAL` data type specifies. For more information, see "Interval Literals" in the *Oracle CEP CQL Language Reference*
object	An opaque data type that can be represented as a `java.lang.Object`.
timestamp	A timestamp `String` in a format that `java.util.Date` specifies. For example: `"Sat, 12 Aug 1995 13:30:00 GMT+0430"`
xmltype	XML values as the Oracle CQL `XMLTYPE` data type specifies. For more information, see "SQL/XML (SQLX)" in the *Oracle CEP CQL Language Reference*.

Example 1–5 shows how to use these types:

Example 1–5   Using com.bea.welvs.ede.api.Type Data Types

```
<wlevs:event-type-repository>
        <wlevs:event-type type-name="SimpleEvent">
            <wlevs:properties>
                <wlevs:property name="msg" type="char" />
                <wlevs:property name="count" type="double" />
                <wlevs:property name="time_stamp" type="timestamp" />
        </wlevs:properties>
    </wlevs:event-type>
...
</wlevs:event-type-repository>
```

For more information, see Section 1.5.4, "How to Create an Oracle CEP Event Type as a Tuple".

1.1.3.3 Event Types for use With a Table Source

When you specify the properties of an event type of a relational database table, you must observe additional JDBC type restrictions. For more information, see:

- Table 10–2, " EPN Assembly File event-type Element Property Attributes"

- Table 10–3, " SQL Column Types and Oracle CEP Type Equivalents"

For more information, see Section 10.3, "Configuring an Oracle CQL Processor Table Source".

1.1.3.4 Event Types for use With the csvgen Adapter

When you specify the properties of an event type for use with the `csvgen` adapter, you may only use the data types that Table 1–2 describes.

Table 1–2   csvgen Adapter Types

Type	Usage
char	Single or multiple character values. Use for both `char` and `java.lang.String` values.
int	Numeric values in the range that `java.lang.Integer` specifies.
long	Numeric values in the range that `java.lang.Long` specifies.
double	Numeric values in the range that `java.lang.Double` specifies.

For more information, see Section 15.4, "Creating a Data Feed File".

1.1.4 Stream Sources and Stream Sinks and Relation Sources and Relation Sinks

All components in an EPN either emit events or receive events. Some components can be both, such as an event bean in the middle of an EPN that receives events from an adapter, for example, and then sends events to a processor.

Oracle CEP models event flow as either streams or relations:

- A stream S is a bag multi-set of elements (s, T) where s is in the schema of S and T is in the time domain. Stream elements are tuple-timestamp pairs, which can be represented as a sequence of timestamped tuple insertions. In other words, a stream is a sequence of timestamped tuples. There could be more than one tuple with the same timestamp. The tuples of an input stream are required to arrive at the system in the order of increasing timestamps.

  You specify a channel as a stream by setting EPN assembly element `wlevs:channel` attribute `is-relation` to `false` (the default).

- A relation is an unordered, time-varying bag of tuples: in other words, an instantaneous relation. At every instant of time, a relation is a bounded set. It can also be represented as a sequence of timestamped tuples that includes insertions, deletions, and updates to capture the changing state of the relation.

  You specify a channel as a realtion by setting EPN assembly element `wlevs:channel` attribute `is-relation` to `true`.

For more information, see:

- Section 9.1.3, "Channels Representing Streams and Relations"

- "Streams and Relations" in the *Oracle CEP CQL Language Reference*

1.1.4.1 Stream and Relation Sources

All components that emit events must implement the
`com.bea.wlevs.ede.api.StreamSource` or
`com.bea.wlevs.ede.api.RelationSource` interface.

The `StreamSource` interface has one method, `setEventSender();` at runtime the
event source component is injected with an
`com.bea.wlevs.ede.api.StreamSender` instance. The `StreamSender` instance,
in turn, has a `sendInsertEvent()` method that the component invokes to actually
send events to the next component in the EPN. The `StreamSender` instance also
provides a `sendHeartbeat()` method. The `sendHeartBeat()` method applies only
to application-timestamped channels.

`RelationSource`, through the `RelationSender`, provides a `sendUpdateEvent()`
and `sendDeleteEvent()` methods.

1.1.4.2 Stream and Relation Sinks

Components that receive events must implement the
`com.bea.wlevs.ede.api.StreamSink` or
`com.bea.wlevs.ede.api.RelationSink` interface.

The `StreamSink` interface has a single callback method, `onInsertEvent()`, that
components implement to receive a list of events from the previous component in the
EPN.

The `RelationSink` interface extends `StreamSink` to model an Oracle CQL relation
sink. The `RelationSink` interface also provides an `onDeleteEvent()` and
`onUpdateEvent()` methods.

1.1.5 Component Configuration Files

Each component in your event processing network (adapter, processor, channel, or
event bean) can have an associated configuration file, although only processors are
required to have a configuration file. The caching system also uses a configuration file,
regardless of whether it is a stage in the event processing network. Component
configuration files in Oracle CEP are XML documents whose structure is defined using
standard XML Schema. You create a single file that contains configuration for all
components in your application, or you can create separate files for each component;
the choice depends on which is easier for you to manage.

The `wlevs_application_config.xsd` schema file describes the structure of
component configuration files. This XSD schema imports the following schemas:

- `wlevs_base_config.xsd`: Defines common elements that are shared between
  application configuration files and the server configuration file

- `wlevs_eventstore_config.xsd`: Defines event store-specific elements.

- `wlevs_diagnostic_config.xsd`: Defines diagnostic elements.

The structure of application configuration files is as follows. There is a top-level root
element named `config` that contains a sequence of sub-elements. Each individual
sub-element contains the configuration data for an Oracle CEP component (processor,
channel, or adapter). For example:

```
<?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>
                <query id="helloworldRule">
                    <![CDATA[ select * from helloworldInputChannel [Now] ]]>
                </query>
            </rules>
        </processor>
        <channel>
            <name>helloworldInputChannel</name>
            <max-size>10000</max-size>
            <max-threads>2</max-threads>
        </channel>
        <channel>
            <name>helloworldOutputChannel</name>
            <max-size>10000</max-size>
            <max-threads>2</max-threads>
        </channel>
</n1:config>
```

For more information, see:

- Section 1.1.6, "Accessing Component and Server Configuration Using the ConfigurationPropertyPlaceholderConfigurer Class"

- Section 1.1.8, "How Components Fit Together"

1.1.6 Accessing Component and Server Configuration Using the ConfigurationPropertyPlaceholderConfigurer Class

Using the `ConfigurationPropertyPlaceholderConfigurer` class, you can reference existing configuration file properties, in both component configuration and server configuration files, using a symbolic placeholder. This allows you to define a value in one place and refer to that one definition rather than hard-coding the same value in many places.

For example, given the element that Example 1–6 shows, you can refer to the `netio` element `port` value as Example 1–7 shows.

Example 1–6   netio Element

```
<netio>
    <name>MyNetIO</name>
    <port>9003</port>
</netio>
```

Example 1–7   Accessing the netio Element port Value

```
<property name="myprop" value="${MyNetIO.port}"/>
```

To use this feature, insert a `ConfigurationPropertyPlaceholderConfigurer` bean in the application context configuration file of your application bundle as Example 1–8 shows.

Example 1–8   Adding a ConfigurationPropertyPlaceholderConfigurer

```
 <bean class="com.bea.wlevs.spring.support.ConfigurationPropertyPlaceholderConfigurer"/>
```

For complete details, see the `com.bea.wlevs.spring.support.ConfigurationPropertyPlaceholderConfigurer` class in the *Oracle CEP Java API Reference*.

1.1.7 EPN Assembly File

You assemble an Oracle CEP application an Event Processing Network (EPN) assembly file before deploying it to the server; this EPN assembly file is an extension of the Spring framework XML configuration file.

For some Oracle CEP features, you specify some configuration in the component configuration file and some in the EPN assembly file.

The `spring-wlevs-v11_0_0_0.xsd` schema file describes the structure of EPN assembly files.

The structure of EPN assembly files is as follows. There is a top-level root element named `beans` that contains a sequence of sub-elements. Each individual sub-element contains the configuration data for an Oracle CEP component. For 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-v11_0_0_0.xsd">

    <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>

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

    <wlevs:channel id="helloworldInputChannel" event-type="HelloWorldEvent" >
        <wlevs:listener ref="helloworldProcessor"/>
        <wlevs:source ref="helloworldAdapter"/>
    </wlevs:channel>

    <wlevs:processor id="helloworldProcessor" />

    <wlevs:channel id="helloworldOutputChannel"
                   event-type="HelloWorldEvent" advertise="true">
        <wlevs:listener>
            <bean class="com.bea.wlevs.example.helloworld.HelloWorldBean"/>
        </wlevs:listener>
        <wlevs:source ref="helloworldProcessor"/>
    </wlevs:channel>

</beans>
```

For more information, see:

- Section 1.4, "Creating the EPN Assembly File"
- Section 1.1.8, "How Components Fit Together"

1.1.8 How Components Fit Together

Oracle CEP applications are made of services that are assembled together to form an Event Processing Network (EPN).

The server uses the Spring framework as its assembly mechanism due to Spring's popularity and simplicity. Oracle CEP has extended the Spring framework to further simplify the process of assembling applications. This approach allows Oracle CEP applications to be easily integrated with existing Spring-beans, and other light-weight programming frameworks that are based upon a dependency injection mechanism.

A common approach for dependency injection is the usage of XML configuration files to declaratively specify the dependencies and assembly of an application. You assemble an Oracle CEP application an EPN assembly file before deploying it to the server; this EPN assembly file is an extension of the Spring framework XML configuration file.

After an application is assembled, it must be package so that it can be deployed into Oracle CEP. This is a simple process. The deployment unit of an application is a plain JAR file, which must contain, at a minimum, the following artifacts:

- The compiled application Java code of the business logic POJO.

- Component configuration files. Each processor is required to have a configuration file, although adapters and streams do not need to have a configuration file if the default configuration is adequate and you do not plan to monitor these components.

- The EPN assembly file.

- A `MANIFEST.MF` file with some additional OSGi entries.

After you assemble the artifacts into a JAR file, you deploy this bundle to Oracle CEP so it can immediately start receiving incoming data.

For more information, see Chapter 14, "Assembling and Deploying Oracle CEP Applications".

1.1.9 Oracle CEP Application Lifecycle

Figure 1–1 shows a state diagram for the Oracle CEP application lifecycle. In this diagram, the state names (`STARTING`, `INITIALIZING`, `RUNNING`, `SUSPENDING`, `SUSPENDED`, and `FAILED`) correspond to the `ApplicationRuntimeMBean` method `getState()` return values. These states are specific to Oracle CEP; they are not OSGi bundle states.

Figure 1–1   Oracle CEP Application Lifecycle State Diagram
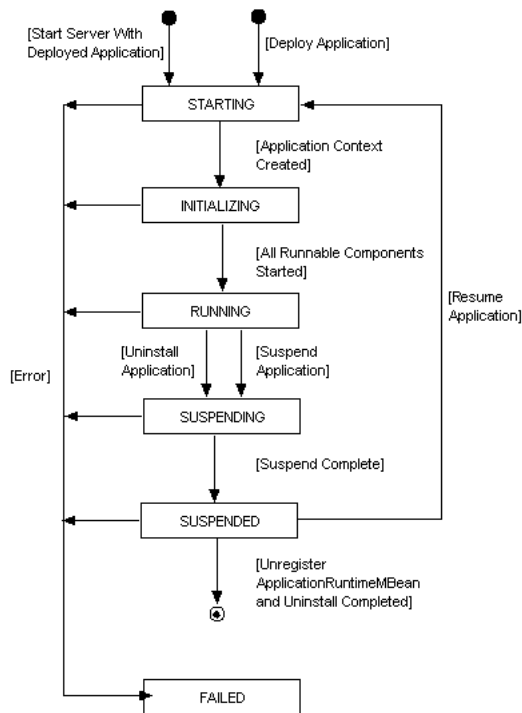
> **Note:**   For information on Oracle CEP server lifecycle, see "Oracle CEP Server Lifecycle" in the *Oracle CEP Administrator's Guide*.

This section describes the lifecycle of an application deployed to the Oracle CEP server and the sequence of `com.bea.wlevs.ede.api` API callbacks. The lifecycle description is broken down into actions that a user performs, including:

- Section 1.1.9.1, "User Action: Installing an Application or Start the Server With Application Already Deployed"

- Section 1.1.9.2, "User Action: Suspend Application"

- Section 1.1.9.3, "User Action: Resume Application"

- Section 1.1.9.4, "User Action: Uninstall Application"

- Section 1.1.9.5, "User Action: Update Application"

This information explains how Oracle CEP manages an application's lifecycle so that you can better use the lifecycle APIs in your application. For a description of these APIs (such as `RunnableBean` and `SuspendableBean`), see:

- Section 1.1.10, "Oracle CEP APIs"

- Appendix G, "Oracle CEP Metadata Annotation Reference"

- *Oracle CEP Java API Reference*

1.1.9.1 User Action: Installing an Application or Start the Server With Application Already Deployed

Oracle CEP performs the following actions:

1. Oracle CEP installs the application as an OSGI bundle. OSGI resolves the imports and exports, and publishes the service.

2. Oracle CEP creates beans (for both standard Spring beans and those that correspond to the Oracle CEP tags in the EPN assembly file). For each bean, Oracle CEP:

   - Sets the properties on the Spring beans. The `<wlevs:instance-property>` values are set on adapters and event-beans.

   - Injects appropriate dependencies into services specified by `@Service` or `@ServiceReference` annotations.

   - Injects appropriate dependencies into static configuration properties.

   - Calls InitializingBean.afterPropertiesSet().

   - Calls configuration callbacks (`@Prepare`,`@Activate`) on Spring beans as well as factory-created stages.

3. Application state is now `INITIALIZING`.

4. Oracle CEP registers the MBeans.

5. Oracle CEP calls `ActivatableBean.afterConfigurationActive()` on all `ActivatableBeans`

6. Oracle CEP calls `ResumableBean.beforeResume()` on all `ResumableBeans`

7. For each bean that implements `RunnableBean`, Oracle CEP starts it running in a thread.

8. Application state is now `RUNNING`.

1.1.9.2 User Action: Suspend Application

Oracle CEP performs the following actions:

1. Oracle CEP calls `SuspendableBean.suspend()`on all `SuspendableBeans`.

2. Application state is now `SUSPENDED`.

1.1.9.3 User Action: Resume Application

Oracle CEP performs the following actions:

1. Oracle CEP calls `ResumableBean.beforeResume()`on all `ResumableBeans`

2. For each bean that implements `RunnableBean`, Oracle CEP starts it running in a thread.

3. Application state is now `RUNNING`.

1.1.9.4 User Action: Uninstall Application

Oracle CEP performs the following actions:

1. Oracle CEP calls `SuspendableBean.suspend()` on all `SuspendableBeans`.

2. Oracle CEP unregisters MBeans.

3. Oracle CEP calls `DisposableBean.dispose()` on all `DisposableBeans`.

4. Oracle CEP uninstalls application bundle from OSGI.

1.1.9.5 User Action: Update Application

This is equivalent to first unistalling an application and then installing it again.

See:

- [Section 1.1.9.4, "User Action: Uninstall Application"](#)

- [Section 1.1.9.1, "User Action: Installing an Application or Start the Server With Application Already Deployed"](#)

1.1.10 Oracle CEP APIs

Oracle CEP provides a variety of Java APIs that you use in your adapter or event bean implementation.

This section describes the APIs in the `com.bea.wlevs.ede.api` package that you will most typically use in your adapters and event beans.

- `Adapter`—Adapters must implement this interface.

- `AdapterFactory`—Adapter factories must implement this interface.

- Component life cycle interfaces—If you want some control over the life cycle of the component you are programming, then your component should implement one or more of the following interfaces:

  - `ActivatableBean`—Use if you want to run some code after all dynamic configuration has been set and the event processing network has been activated. Implement the `afterConfigurationActive()` method.

  - `DisposableBean`—Use if you want to release resources when the application is undeployed. Implement the `destroy()` method in your component code.

  - `InitializingBean`—Use if you require custom initialization after Oracle CEP has set all the properties of the component. Implement the `afterPropertiesSet()` method.

  - `ResumableBean`—Use if you want to perform some task, such as acquire or configure resources, before the component resumes work.

  - `RunnableBean`—Use if you want the component to be run in a thread.

    The Spring framework implements similar bean life cycle interfaces; however, the equivalent Spring interfaces do not allow you to manipulate beans that were created by factories, while the Oracle CEP interfaces do.

  - `SuspendableBean`—Use if you want to suspend resources or stop processing events when the event processing network is suspended. Implement the `suspend()` method.

  See also [Appendix G, "Oracle CEP Metadata Annotation Reference"](#) for additional lifecycle annotations.

- `EventBuilder`—Use to create events whose Java representation does not expose the necessary setter and getter methods for its properties. If your event type is represented with a JavaBean with all required getter and setter methods, then you do not need to create an `EventBuilder`.

- `EventBuilder.Factory`—Factory for creating `EventBuilders`.

- `StreamSink`—Components that want to receive events as an Oracle CEP stream must implement this interface. An Oracle CEP stream has the following characteristics:

  - Append-only, that is, events are always appended to the end of the stream.

  - Unbounded and generally need a window to be defined before it can be processed.

- Events have non-decreasing time-stamps.

  The interface has a callback method, `onInsertEvent()`, in which programmers put the code that handles the received events.

- `StreamSource`—Components that send events modeling an Oracle CEP stream, such as adapters, must implement this interface. The interface has a `setEventSender()` method for setting the `StreamSender`, which actually sends the event to the next component in the network.

- `RelationSink`—Components that want to receive events modeling an Oracle CEP relation must implement this interface. An Oracle CEP relation has the following characteristics:

  - Supports events that insert, delete, and update its content.

  - Is always known at an instant time.

  - Events have non-decreasing time-stamps.

  The interface has callback methods `onDeleteEvent()` and `onInsertEvent()` in which programmers put the code that handles event deletion and event insertion.

- `RelationSource`—Components that send events modeling an Oracle CEP relation, such as adapters, must implement this interface. The interface has a `setEventSender()` method for setting the `RelationSender`, which actually sends the event to the next component in the network.

For more information, see:

- *Oracle CEP Java API Reference* for the full reference documentation for all classes and interfaces.

- Chapter 8, "Configuring Custom Adapters, Event Beans, and Spring Beans" as well as the HelloWorld and FX examples in the installed product, for sample Java code that uses these APIs.

- Section 1.6, "Configuring Oracle CEP Resource Access" for information on using Oracle CEP annotations and deployment XML to configure resource injection.

1.2 Oracle CEP IDE for Eclipse

Oracle provides an IDE targeted specifically to programmers that want to develop Oracle CEP applications. Oracle CEP IDE for Eclipse is a set of plugins for the Eclipse IDE designed to help develop, deploy, and debug applications for Oracle CEP.

The key features of the Oracle CEP IDE for Eclipse are as follows:

- Project creation wizards and templates to quickly get started building event driven applications.

- Advanced editors for source files including Java and XML files common to Oracle CEP applications.

- Integrated server management to seamlessly start, stop, and deploy to Oracle CEP server instances all from within the IDE.

- Integrated debugging.

- Event Processing Network (EPN) visual design views for orienting and navigating in event processing applications.

■ Integrated support for the Oracle CEP Visualizer so you can use the Oracle CEP Visualizer from within the IDE.

Although it is not required or assumed that you are using the Oracle CEP IDE, Oracle recommends that you give it a try.

For detailed instructions on installing and using the IDE, see Chapter 2, "Overview of the Oracle CEP IDE for Eclipse".

1.3 Creating an Oracle CEP Application

The following procedure shows the *suggested* start-to-finish steps to create an Oracle CEP application. Although it is not required to program and configure the various components in the order shown, the procedure shows a typical and logical flow recommended by Oracle.

It is assumed in the procedure that you are using an IDE, although it is not required and the one you use is your choice. For one targeted to Oracle CEP developers, see Section 1.2, "Oracle CEP IDE for Eclipse."

To create an Oracle CEP application:

1. Set up your environment as described in "Setting Your Development Environment" in the *Oracle CEP Getting Started*.

2. Design your event processing network (EPN).

   This step involves creating the EPN assembly file, adding the full list of components that make up the application and how they are connected to each other, as well as registering the event types used in your application.

   This step combines both designing of your application, in particular determining the components that you need to configure and code, as well as creating the actual XML file that specifies all the components. You will likely be constantly updating this XML file as you implement your application, but Oracle recommends you start with this step so you have a high-level view of your application.

   For details, see Section 1.4, "Creating the EPN Assembly File."

3. Design the rules that the processors are going to use to select events from the channel.

   See:

   ■ *Oracle CEP CQL Language Reference*

   ■ *Oracle CEP EPL Language Reference*

   Note: Oracle CQL replaces Event Processing Language (EPL) in Oracle CEP Release 11gR1 (11.1.1). Oracle CEP supports EPL for backwards compatibility.

   ---

4. Determine the event types that your application is going to use, and, if creating your own JavaBean, program the Java file.

   See Section 1.5, "Creating Oracle CEP Event Types."

5. Program, and optionally configure, the adapters or event beans that act as inbound, intermediate, or outbound components of your event processing network. You can create your own adapters or event beans, or use the adapters provided by Oracle CEP. For details, see:

- Section 1.1.10, "Oracle CEP APIs"
- Chapter 8, "Configuring Custom Adapters, Event Beans, and Spring Beans"
- Chapter 6, "Configuring JMS Adapters"
- Chapter 7, "Configuring HTTP Publish-Subscribe Server Adapters"
- Section 1.6, "Configuring Oracle CEP Resource Access"

6. Configure the processors by creating their component configuration XML files; the most important part of this step is designing and declaring the initial rules that are associated with each processor.

   See:

   - Chapter 10, "Configuring Oracle CQL Processors"
   - Chapter 11, "Configuring EPL Processors"

7. Optionally configure the channels that stream data between adapters, processors, and the business logic POJO by creating their configuration XML files.

   See Chapter 9, "Configuring Channels."

8. Optionally configure the caching system to publish or consume events to and from a cache to increase the availability of the events and increase the performance of your applications.

   See Chapter 12, "Configuring Caching."

9. Optionally, use the Oracle CEP server log subsystem to write log messages from your application to the Oracle CEP server log:

```
import org.apache.commons.logging.Log;
import org.apache.commons.logging.LogFactory;
...
Log LOG=LogFactory.getLog("LogName");
...
LOG.debug("Some debug information");
...
```

   Using the Oracle CEP Visualizer, you can deploy your application, configure the log level for your application, and view the Oracle CEP server console.

   For more information, see:

   - "Configuring Logging and Debugging for Oracle CEP" in the *Oracle CEP Administrator's Guide*
   - "How to Configure Component Logging" in the *Oracle CEP Visualizer User's Guide*
   - "How to View Console Output" in the *Oracle CEP Visualizer User's Guide*

See Section 1.7, "Next Steps" for the list of steps you should follow after you have completed programming your application, such as packaging and deploying.

1.4 Creating the EPN Assembly File

You use the Event Processing Network (EPN) assembly file to declare the components that make up your Oracle CEP application and how they are connected to each other. You also use the file to register event types of your application, as well as the Java classes that implement the adapter and POJO components of your application.

For an example of an EPN assembly file, see the "Foreign Exchange (FX) Example" in the *Oracle CEP Getting Started*. For additional information about Spring and OSGi, see Appendix A, "Additional Information about Spring and OSGi."

1.4.1 How to Create the EPN Assembly File Using Oracle CEP IDE for Eclipse

The most efficient and least error-prone way to create and edit the EPN file is using the EPN editor in the Oracle CEP IDE for Eclipse. Optionally, you can create a channel configuration file manually (see Section 1.4.2, "How to Create the EPN Assembly File Manually").

For more information, see Section 5.2, "EPN Editor Overview".

To create the EPN assembly file using Oracle CEP IDE for Eclipse:

1. Create an Oracle CEP project.

See Section 3.2, "Creating Oracle CEP Projects".

2. Open the EPN editor.

See Section 5.1, "Opening the EPN Editor".

3. Create and connect nodes on the EPN.

See Section 5.4, "Using the EPN Editor".

1.4.2 How to Create the EPN Assembly File Manually

Although the Oracle CEP IDE for Eclipse EPN editor is the most efficient and least error-prone way to create and edit the EPN file (see Section 1.4.1, "How to Create the EPN Assembly File Using Oracle CEP IDE for Eclipse"), alternatively, you can also create and maintain the EPN file manually.

As is often true with Spring, there are different ways to use the tags to define your event network. This section shows one way.

See Section B.2, "EPN Assembly Schema spring-wlevs-v11_0_0_0.xsd" for the full reference information on the other tags and attributes you can use.

To create the EPN assembly file manually:

1. Using your favorite XML or plain text editor, create an XML file with the `<beans>` root element and namespace declarations as follows:

```
<?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">
...
</beans>
```

If you are not going to use any of the Spring-OSGI tags in the XML file, then their corresponding namespace declarations, shown in bold in the preceding example, are not required.

2. If you have programmed an adapter factory, add an `<osgi:service ...>` Spring tag to register the factory as an OSGi service. For example:

```
<osgi:service interface="com.bea.wlevs.ede.api.AdapterFactory">
    <osgi:service-properties>
        <entry key="type" value="hellomsgs<"/entry>
    </osgi:service-properties>
    <bean
class="com.bea.wlevs.adapter.example.helloworld.HelloWorldAdapterFactory" />
</osgi:service>
```

Specify the Oracle CEP-provided adapter factory (`com.bea.wlevs.ede.api.AdapterFactory`) for the `interface` attribute. Use the `<osgi-service-properties>` tag to give the OSGI service a type name, in the example above the name is `hellomsgs`; you will reference this label later when you declare the adapter components of your application. Finally, use the `<bean>` Spring tag to register the your adapter factory bean in the Spring application context; this class generates instances of the adapter.

> **Caution:** Be sure the type name (`hellomsgs` in the preceding example) is unique across all applications deployed to a particular Oracle CEP. The OSGI service registry is per server, not per application, so if two different adapter factory services have been registered with the same type name, it is undefined which adapter factory a particular application will use. To avoid this confusion, be sure that the value of the `<entry key="type" ...>` entry for each OSGI-registered adapter factory in each EPN assembly file for a server is unique.

3. Add a `<wlevs:event-type-repository>` tag to register the event types that you use throughout your application, such as in the adapter implementations, business logic POJO, and the EPL rules associated with the processor components. For each event type in your application, add a `<wlevs:event-type>` child tag.

Event types are simple JavaBeans that you either code yourself (recommended) or let Oracle CEP automatically generate from the meta data you provide in the `<wlevs:event-type>` tag. If you code the JavaBean yourself, use a `<wlevs:class>` tag to specify your JavaBean class. You can optionally use the `<wlevs:property name="builderFactory">` tag to specify the Spring bean that acts as a builder factory for the event type, if you have programmed a factory. If you want Oracle CEP to automatically generate the JavaBean class, use the `<wlevs:property>` tag to list each property of the event type. The following example is taken from the FX sample:

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

```
    </wlevs:event-type>
</wlevs:event-type-repository>
```

See Section D.12, "wlevs:event-type-repository" for reference information about this tag.

See Section 1.5, "Creating Oracle CEP Event Types" for additional information about creating event types.

4. For each adapter component in your application, add a `<wlevs:adapter>` tag to declare that the component is part of the event processing network. Use the required `id` attribute to give it a unique ID and the `provider` attribute to specify the type of data feed to which the adapter will be listening. Use the `<wlevs:instance-property>` child tag to pass the adapter the properties it expects. For example, the `csvgen` adapter, provided by Oracle CEP to test your EPL rules with a simulated data feed, defines a `setPort()` method and thus expects a `port` property, among other properties. Use the `provider` attribute to specify the adapter factory, typically registered as an OSGi service; you can also use the `csvgen` keyword to specify the `csvgen` adapter.

The following example declares the `helloWorldAdapter` of the HelloWorld example:

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

In the example, the property `message` is passed to the adapter. The adapter factory provider is `hellomsgs`, which refers to the type name of the adapter factory OSGI service. The `advertise` attribute, common to all components, enables monitoring for the adapter; by default, manageability of the component is disabled due to possible performance impacts.

See Section D.2, "wlevs:adapter" for reference information about this tag, in particular additional optional attributes and child tags.

5. For each processor component in your application, add a `<wlevs:processor>` tag. Use the `id` attribute to give it a unique ID. Use either the `listeners` attribute or `<wlevs:listener>` child tag to specify the components that listen to the processor. The following two examples are equivalent:

```
<wlevs:processor id="preprocessorAmer" listeners="spreaderIn"/>
<wlevs:processor id="preprocessorAmer">
        <wlevs:listener ref="spreaderIn"/>
</wlevs:processor>
```

In the examples, the `spreaderIn` channel component listens to the `preprocessorAmer` processor.

See Section D.20, "wlevs:processor" for reference information about this tag, in particular additional optional attributes, such as `advertise` for enabling monitoring of the component.

6. For each channel component in your application, add a `<wlevs:channel>` tag to declare that the component is part of the event processing network. Use the `id` attribute to give it a unique ID. Use the `<wlevs:listener>` and `<wlevs:source>` child tags to specify the components that act as listeners and sources for the channel. For example:

```
<wlevs:channel id="fxMarketAmerOut">
```

```
        <wlevs:listener ref="preprocessorAmer"/>
        <wlevs:source ref="fxMarketAmer"/>
</wlevs:channel>
```

In the example, the `fxMarketAmerOut` channel listens to the `fxMarketAmer` component, and the `preprocessorAmer` component in turn listens to the `fxMarketAmerOut` channel.

Nest the declaration of the business logic POJO, called `outputBean` in the example, using a standard Spring `<bean>` tag inside a `<wlevs:listener>` tag, as shown:

```
<wlevs:channel id="spreaderOut" advertise="true">
    <wlevs:listener>
       <!-- Create business object -->
        <bean id="outputBean"
              class="com.bea.wlevs.example.fx.OutputBean"
              autowire="byName"/>
    </wlevs:listener>
</wlevs:channel>
```

The `advertise` attribute is common to all Oracle CEP application assembly tags and is used to register the component as a service in the OSGI registry.

Section D.10, "wlevs:channel" for reference information about this tag, in particular additional optional attributes, such as `advertise` for enabling monitoring of the component.

1.5 Creating Oracle CEP Event Types

Event types define the properties of the events that are handled by Oracle CEP applications. Adapters receive incoming events from different event sources, such as JMS, or financial market data feeds. You must define these events by an event type before a processor is able to handle them. You then use these event types in the adapter and POJO Java code, as well as in the Oracle CQL and EPL rules you associate with the processors.

This section describes:

- Section 1.5.1, "How to Create an Oracle CEP Event Type as a JavaBean"

- Section 1.5.2, "How to Create an Oracle CEP Event Type as a Java Class"

- Section 1.5.3, "How to Create an Oracle CEP Event Type as a java.util.Map"

- Section 1.5.4, "How to Create an Oracle CEP Event Type as a Tuple"

- Section 1.5.5, "Using an Event Type Builder Factory"

- Section 1.5.6, "Sharing Event Types Between Application Bundles"

For more information, see Section 1.1.2, "Event Types".

1.5.1 How to Create an Oracle CEP Event Type as a JavaBean

This procedure describes how to create and register an Oracle CEP event type as a Java Bean. This is the preferred approach.

See the JavaBeans Tutorial at
http://java.sun.com/docs/books/tutorial/javabeans/ for additional details.

For more information on valid event type data types, see Section 1.1.3.1, "Event Types Specified as Java Bean, Java Class, or java.util.Map".

To create an Oracle CEP event type as a Java bean:

1. Create a Java Bean class to represent your event type.

   Follow standard JavaBeans programming guidelines. See the JavaBeans Tutorial at http://java.sun.com/docs/books/tutorial/javabeans/ for additional details.

   Oracle recommends that, if possible, you make your event type JavaBeans immutable to improve performance. Immutable beans help the garbage collection work much better. Immutable beans are read only (only getters) and have public constructors with arguments that satisfy immutability.

   Example 1–9 shows the MarketEvent which is implemented by the com.bea.wlevs.example.algotrading.event.MarketEvent class.

Example 1–9   MarketEvent Class

```
package com.bea.wlevs.example.algotrading.event;

import java.util.Date;

public final class MarketEvent {
    private final Long timestamp;

    private final String symbol;

    private final Double price;

    private final Long volume;

    private final Long latencyTimestamp;

    public MarketEvent(final Long timestamp, final String symbol,
            final Double price, final Long volume, final Long latencyTimestamp) {
        this.timestamp = timestamp;
        this.symbol = symbol;
        this.price = price;
        this.volume = volume;
        this.latencyTimestamp = latencyTimestamp;
    }

    public Double getPrice() {
        return price;
    }

    public String getSymbol() {
        return symbol;
    }

    public Long getTimestamp() {
        return timestamp;
    }

    public Long getLatencyTimestamp() {
        return latencyTimestamp;
    }

    public Long getVolume() {
        return volume;
    }
```

```
}
```

2. Compile the JavaBean that represents your event type.

3. Register your JavaBean event type in the Oracle CEP event type repository:

   a. To register declaratively, edit the EPN assembly file using the `wlevs:event-type-repository` element `wlevs:event-type` child element as Example 1–10 shows.

Example 1–10   EPN Assembly File event-type-repository

```
<wlevs:event-type-repository>
    <wlevs:event-type type-name="MarketEvent">
      <wlevs:class>
          com.bea.wlevs.example.algotrading.event.MarketEvent
      </wlevs:class>
    </wlevs:event-type>
</wlevs:event-type-repository>
```

   b. To register programatically, use the `EventTypeRepository` class as Example 1–11 shows.

Example 1–11   Programmatically Registering an Event

```
EventTypeRepository rep = getEventTypeRepository();
rep.registerEventType(
    "MarketEvent",
    com.bea.wlevs.example.algotrading.event.MarketEvent.getClass()
);
```

For more information, see the *Oracle CEP Java API Reference*.

4. Use the event type:

   ■ Reference the event types as standard JavaBeans in the Java code of the adapters and business logic POJO in your application.

```
public void onEvent(List newEvents)
        throws RejectEventException {
    for (Object event : newEvents) {
            MarketEvent marketEvent = (MarketEvent) event;
        System.out.println("Price: " + marketEvent.getPrice());
    }
}
```

   ■ Access the event types from Oracle CQL and EPL rules:

   The following Oracle CQL rule shows how you can reference the `MarketEvent` in a `SELECT` statement:

```
<query id="helloworldRule">
    <![CDATA[ select MarketEvent.price from marketEventChannel [Now] ]]>
</query>
```

1.5.2 How to Create an Oracle CEP Event Type as a Java Class

This procedure describes how to create and register an Oracle CEP event type as a Java class that does not conform to the JavaBeans programming guidelines.

For more information on valid event type data types, see Section 1.1.3.1, "Event Types Specified as Java Bean, Java Class, or java.util.Map".

To create an Oracle CEP event type as a Java class:

1. Create a Java class to represent your event type.

   Oracle recommends that, if possible, you make your event type Java class immutable to improve performance. Immutable beans help the garbage collection work much better. Immutable beans are read only (only getters) and have public constructors with arguments that satisfy immutability.

   Example 1–9 shows the `ForeignExchangeEvent` which is implemented by the `ForeignExchangeEvent` inner class of `com.bea.wlevs.example.fx.OutputBean`.

Example 1–12   ForeignExchangeEvent Class: Does not Conform to JavaBean Standards

```
package com.bea.wlevs.example.fx.OutputBean.ForeignExchangeEvent;

static public class ForeignExchangeEvent {
    private String symbol;
    private Double price;
    private String fromRate;
    private String toRate;
    private Long clientTimestamp;
    private Long adapterTimestamp;

    public ForeignExchangeEvent() {

    }

    public ForeignExchangeEvent(String symbol, Double price, String fromRate, String toRate)
{
        this.symbol = symbol;
        this.price = price;
        this.fromRate = fromRate;
        this.toRate = toRate;

        if (clientTimestamp == null)
            this.clientTimestamp = new Long(0);

        if (adapterTimestamp == null)
            this.adapterTimestamp = new Long(0);
    }

    public String getSymbol() {
        return symbol;
    }

    public Double getPrice() {
        return price;
    }

    public String getFromRate() {
        return fromRate;
    }

    public String getToRate() {
        return toRate;
    }

    public Long getClientTimestamp() {
        return clientTimestamp;
    }

    public Long getAdapterTimestamp() {
        return adapterTimestamp;
    }
```

```
}
```

2. Create a Java class to represent your event type builder as Example 1–13 shows.

Example 1–13   ForeignExchangeBuilderFactory

```
package com.bea.wlevs.example.fx;

import java.util.HashMap;
import java.util.Map;

import com.bea.wlevs.ede.api.EventBuilder;
import com.bea.wlevs.example.fx.OutputBean.ForeignExchangeEvent;

public class ForeignExchangeBuilderFactory implements EventBuilder.Factory {

    public EventBuilder createBuilder() {
        return new ForeignExchangeBuilder();
    }

    static class ForeignExchangeBuilder implements EventBuilder {
        private Map<String,Object> values = new HashMap<String,Object>(10);

        public Object createEvent() {
            return new ForeignExchangeEvent(
                (String) values.get("symbol"),
                (Double) values.get("price"),
                (String) values.get("fromRate"),
                (String) values.get("toRate"));
        }

        public void put(String property, Object value) throws IllegalStateException {
            values.put(property, value);
        }

    }
}
```

Oracle CEP uses the event builder factory to create event instances that do not follow the JavaBean specification. See Section 1.5.5, "Using an Event Type Builder Factory" for additional information about using an event type builder factory.

3. Compile the Java classes that represent your event type and event builder factory.

4. Register your event type and event builder factory in the Oracle CEP event type repository:

    a. To register declaratively, edit the EPN assembly file using the `wlevs:event-type-repository` element `wlevs:event-type` child element as Example 1–10 shows.

    Use the `<wlevs:class>` tag to point to your JavaBean class, and then use the `<wlevs:property name="builderFactory">` tag to specify a custom event builder factory for the event type.

Example 1–14   EPN Assembly File event-type-repository

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

```
            class="com.bea.wlevs.example.fx.ForeignExchangeBuilderFactory"/>
        </wlevs:property>
        </wlevs:event-type>
    </wlevs:event-type-repository>
```

> **b.** To register programatically, use the `EventTypeRepository` class as
> Example 1–11 shows.

Example 1–15 Programmatically Registering an Event

```
EventTypeRepository rep = getEventTypeRepository();
ForeignExchangeBuilderFactory factory = new ForeignExchangeBuilderFactory();
rep.registerEventType(
    "ForeignExchangeEvent",
    com.bea.wlevs.example.fx.OutputBean.ForeignExchangeEvent.getClass(),
    factory.createBuilder()
);
```

> For more information, see the *Oracle CEP Java API Reference*.

5. Use the event type:

- Reference the event types as standard JavaBeans in the Java code of the adapters and business logic POJO in your application.

```
public void onEvent(List newEvents)
        throws RejectEventException {
    for (Object event : newEvents) {
            ForeignExchangeEvent fxEvent = (ForeignExchangeEvent) event;
        System.out.println("Price: " + fxEvent.getPrice());
    }
}
```

- Access the event types from Oracle CQL and EPL rules:

  The following Oracle CQL rule shows how you can reference the `MarketEvent` in a `SELECT` statement:

```
<query id="helloworldRule">
    <![CDATA[ select ForeignExchangeEvent.price from marketEventChannel
[Now] ]]>
</query>
```

1.5.3 How to Create an Oracle CEP Event Type as a java.util.Map

This procedure describes how to create and register an Oracle CEP event type as a `java.util.Map`. Oracle recommends that you create an Oracle CEP event type as a JavaBean (see Section 1.5.1, "How to Create an Oracle CEP Event Type as a JavaBean").

For more information on valid event type data types, see Section 1.1.3.1, "Event Types Specified as Java Bean, Java Class, or java.util.Map".

To create an Oracle CEP event type as a java.util.Map:

1. Decide on the properties your event type requires.

2. Register your event type in the Oracle CEP event type repository:

> **a.** To register declaratively, edit the EPN assembly file using the `wlevs:event-type-repository` element `wlevs:event-type` child element as Example 1–16 shows.

Example 1–16 EPN Assembly File event-type-repository

```
<wlevs:event-type-repository>
    <wlevs:event-type type-name="AnotherEvent">
        <wlevs:property>
            <entry key="name" value="java.lang.String"/>
            <entry key="age" value="java.lang.Integer"/>
            <entry key="address" value="java.lang.String"/>
        </wlevs:property>
    </wlevs:event-type>
</wlevs:event-type-repository>
```

At runtime, Oracle CEP generates a bean instance of `AnotherEvent` for you. The `AnotherEvent` has three properties: `name`, `age`, and `address`.

b. To register programatically, use the `EventTypeRepository` class as Example 1–17 shows.

Example 1–17 Programmatically Registering an Event

```
EventTypeRepository rep = getEventTypeRepository();
java.util.Map map = new Map({name, java.lang.String}, {age, java.lang.Integer}, {address,
java.lang.String});
rep.registerEventType(
    "AnotherEvent",
    map
);
```

For more information, see the *Oracle CEP Java API Reference*.

3. Use the event type:

- Reference the event types as standard JavaBeans in the Java code of the adapters and business logic POJO in your application.

```
public void onEvent(List newEvents)
        throws RejectEventException {
    for (Object event : newEvents) {
            AnotherEvent anEvent = (AnotherEvent) event;
        System.out.println("Age: " + anEvent.getAge());
    }
}
```

- Access the event types from Oracle CQL and EPL rules:

  The following Oracle CQL rule shows how you can reference the `MarketEvent` in a `SELECT` statement:

```
<query id="helloworldRule">
    <![CDATA[ select AnotherEvent.age from eventChannel [Now] ]]>
</query>
```

1.5.4 How to Create an Oracle CEP Event Type as a Tuple

This procedure describes how to create and register an Oracle CEP event type declaratively in the EPN assembly file as a tuple. Oracle recommends that you create an Oracle CEP event type as a JavaBean (see Section 1.5.1, "How to Create an Oracle CEP Event Type as a JavaBean").

For more information on valid event type data types, see Section 1.1.3.2, "Event Types Specified as a Tuple".

To create an Oracle CEP event type as a tuple:

1. Decide on the properties your event type requires.

2. Register your event type declaratively in the Oracle CEP event type repository:

   To register declaratively, edit the EPN assembly file using the `wlevs:event-type-repository` element `wlevs:event-type` child element as Example 1–18 shows.

Example 1–18   EPN Assembly File event-type-repository

```
<wlevs:event-type-repository>
    <wlevs:event-type type-name="CrossRateEvent">
        <wlevs:properties>
            <wlevs:property name="price" type="double"/>
            <wlevs:property name="fromRate" type="char"/>
            <wlevs:property name="toRate" type="char"/>
        </wlevs:properties>
    </wlevs:event-type>
</wlevs:event-type-repository>
```

At runtime, Oracle CEP generates a bean instance of `CrossRateEvent` for you. The `CrossRateEvent` has three properties: `price`, `fromRate`, and `toRate`.

For more information on the valid values of the `type` attribute, see Section 1.1.3.2, "Event Types Specified as a Tuple".

3. Use the event type:

   - Reference the event types as standard JavaBeans in the Java code of the adapters and business logic POJO in your application.

   ```
   public void onEvent(List newEvents)
           throws RejectEventException {
       for (Object event : newEvents) {
               CrossRateEvent anEvent = (CrossRateEvent) event;
           System.out.println("Price: " + anEvent.getPrice());
       }
   }
   ```

   - Access the event types from Oracle CQL and EPL rules:

   The following Oracle CQL rule shows how you can reference the `MarketEvent` in a `SELECT` statement:

   ```
   <query id="helloworldRule">
       <![CDATA[ select CrossRateEvent.price from eventChannel [Now] ]]>
   </query>
   ```

1.5.5 Using an Event Type Builder Factory

When you register an event type in the repository using the `wlevs:event-type` tag, Oracle CEP creates instances of the event using the information in the event type class.

Sometimes, however, you might want or need to have more control over how the event type instances are created. This is required if the POJO that represents the event does not expose the necessary getter and setter methods for the event properties. For example, assume the event type has a `firstname` property, but the EPL rule that executes on the event type assumes the property is called `fname`. Further assume that you cannot change either the event type class (because you are using a shared event class from another bundle, for example) or the EPL rule to make them compatible with each other. In this case you can use an event type builder factory to change the way the

event type instance is created so that the property is named `fname` rather than `firstname`.

When you program the event type builder factory, you must implement the `EventBuilder.Factory` inner interface of the `com.bea.wlevs.ede.api.EventBuilder` interface; see the *Oracle CEP Java API Reference* for details about the methods you must implement, such as `createBuilder()` and `createEvent()`.

The following example of an event type builder factory class is taken from the FX sample:

```
package com.bea.wlevs.example.fx;
import java.util.HashMap;
import java.util.Map;
import com.bea.wlevs.ede.api.EventBuilder;
import com.bea.wlevs.example.fx.OutputBean.ForeignExchangeEvent;
public class ForeignExchangeBuilderFactory implements EventBuilder.Factory {
        public EventBuilder createBuilder() {
                return new ForeignExchangeBuilder();
        }
        static class ForeignExchangeBuilder implements EventBuilder {
                private Map<String,Object> values = new HashMap<String,Object>(10);
                public Object createEvent() {
                        return new ForeignExchangeEvent(
                                (String) values.get("symbol"),
                                (Double) values.get("price"),
                                (String) values.get("fromRate"),
                                (String) values.get("toRate"));
                }
                public void put(String property, Object value) throws IllegalStateException {
                        values.put(property, value);
                }
        }
}
```

When you register the event type in the EPN assembly file, use the `<wlevs:property name="builderFactory">` child tag of the `<wlevs:event-type>` tag to specify the name of the factory class. The hard-coded `builderFactory` value of the `name` attribute alerts Oracle CEP that it should use the specified factory class, rather than its own default factory, when creating instances of this event. For example, in the FX example, the builder factory is registered as shown in bold:

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

1.5.6 Sharing Event Types Between Application Bundles

Each Oracle CEP application gets its own Java classloader and loads application classes using that classloader. This means that, by default, one application cannot access the classes in another application. However, because the event type repository is a singleton service, you can configure the repository in one bundle and then explicitly export the event type classes so that applications in separate bundles (deployed to the same Oracle CEP server) can use these shared event types.

The event type names in this case are scoped to the entire Oracle CEP server instance. This means that you will get an exception if you try to create an event type that has the same name as an event type that has been shared from another bundle, but the event type classes are different.

To share event type classes, add their package name to the `Export-Package` header of the `MANIFEST.MF` file of the bundle that contains the event type repository you want to share.

Be sure you deploy the bundle that contains the event type repository *before* all bundles that contain applications that use the shared event types, or you will get a deployment exception.

1.6 Configuring Oracle CEP Resource Access

By using Oracle CEP and standard Java annotations and deployment XML, you can configure the Oracle CEP Spring container to inject resources (such as data sources or persistence managers, and so on) into your Oracle CEP application components.

The Spring container typically injects resources during component initialization. However, it can also inject and re-inject resources at runtime and supports the use of JNDI lookups at runtime.

Oracle CEP supports the following types of resource access:

- Section 1.6.1, "Static Resource Injection"
- Section 1.6.2, "Dynamic Resource Injection"
- Section 1.6.3, "Dynamic Resource Lookup Using JNDI"

See Section 1.6.4, "Understanding Resource Name Resolution" for information on resource name resolution.

See Appendix G, "Oracle CEP Metadata Annotation Reference" for complete details of all Oracle CEP annotations.

In the following sections, consider the example resource that Example 1–19 shows. This is a data source resource named `StockDS` that you specifiy in the Oracle CEP server `config.xml` file.

Example 1–19   Sample Resource: Data Source StockDS

```
<config ...>
    <data-source>
        <name>StockDs</name>
        ...
        <driver-params>
            <url>jdbc:derby:</url>
            ...
        <driver-params>
    </data-source>
...
</config>
```

1.6.1 Static Resource Injection

Static resource injection refers to the injection of resources during the initialization phase of the component lifecycle. Once injected, resources are fixed, or static, while the component is active or running.

You can configure static resource injection using:

- Section 1.6.1.1, "Static Resource Names"

- Section 1.6.1.2, "Dynamic Resource Names"

1.6.1.1 Static Resource Names

When you configure static resource injection using static resource names, the resource name you use in the `@Resource` annotation or Oracle CEP assembly XML file must exactly match the name of the resource as you defined it. The resource name is static in the sense that you cannot change it without recompiling.

To configure static resource injection using static resource names at design time, you use the standard `javax.annotation.Resource` annotation as Example 1–20 shows.

To override design time configuration at deploy time, you use Oracle CEP assembly file XML as Example 1–21 shows.

In Example 1–20 and Example 1–21, the resource name `StockDs` exactly matches the name of the data source in the Oracle CEP server `config.xml` file as Example 1–19 shows.

Example 1–20   Static Resource Injection Using Static Resource Names: Annotations

```
import javax.annotation.Resource;

public class SimpleBean implements EventBean {
...
    @Resource (name="StockDs")
    public void setDataSource (DataSource dataSource){
        this.dataSource = dataSource;
    }
}
```

Example 1–21   Static Resource Injection Using Static Resource Names: XML

```
< wlevs:event-bean id="simpleBean" class="...SimpleBean"/>
    <wlevs:resource property="dataSource" name="StockDs"/>
</wlevs:event-bean>
```

If the name of the EventBean set method matches the name of the resource, then the `@Resource` annotation `name` attribute is not needed as Example 1–22 shows. Similarly, in this case, the `wlevs:resource` element `name` attribute is not needed as Example 1–23.

Example 1–22   Static Resource Injection Using Static Resource Names: Annotations

```
import javax.annotation.Resource;

public class SimpleBean implements EventBean {
...
    @Resource ()
    public void setStockDs (DataSource dataSource){
        this.dataSource = dataSource;
    }
}
```

Example 1–23   Static Resource Injection Using Static Resource Names: XML

```
< wlevs:event-bean id="simpleBean" class="...SimpleBean"/>
    <wlevs:resource property="dataSource"/>
</wlevs:event-bean>
```

1.6.1.2 Dynamic Resource Names

A dynamic resource name is one that is specified as part of the dynamic or external configuration of an application. Using a dynamic resource name, the deployer or administrator can change the resource name without requiring that the application developer modify the application code or the Spring application context.

To add a dynamic resource name to a component, such as an adapter or POJO, you must first specify custom configuration for your component that contains the resource name as Example 1–24 shows.

Example 1–24   Custom Component Configuration

```
<simple-bean>
    <name>SimpleBean</name>
    <trade-datasource>StockDs</trade-datasource>
</simple-bean>
```

To configure static resource injection using dynamic resource names at design time, you use the standard `javax.annotation.Resource` annotation as Example 1–25 shows.

To override design time configuration at deploy time, you use Oracle CEP assembly file XML as Example 1–26 shows.

Example 1–25   Static Resource Injection Using Dynamic Resource Names: Annotations

```
import javax.annotation.Resource;

public class SimpleBean implements EventBean {
...

    @Resource (name="trade-datasource")
    public void setDataSource (DataSource dataSource){
        this.dataSource = dataSource;
    }
}
```

Example 1–26   Static Resource Injection Using Dynamic Resource Names: XML

```
< wlevs:event-bean id="simpleBean" class="...SimpleBean"/>
    <wlevs:resource property="dataSource" name="trade-datasource"/>
</wlevs:event-bean>
```

1.6.2 Dynamic Resource Injection

Dynamic resource injection refers to the injection of resources dynamically while the component is active in response to a dynamic configuration change using Spring container method injection.

To configure dynamic resource injection at design time, you use the standard `javax.annotation.Resource` annotation as Example 1–27 shows.

Example 1–27   Dynamic Resource Injection: Annotations

```
import javax.annotations.Resource;

public class SimpleBean implements EventBean {
...
    @Resource ("trade-datasource")
    public abstract DataSource getDataSource ();
    ...
    }
```

The component calls the getDataSource() method at run time whenever it needs to retrieve a new instance of the resource that the resource name trade-datasource refers to.

Typically, the component calls the getDataSource() method during the @Prepare or @Activate methods when dynamic configuration changes are handled. For more information see:

- Section G.2, "com.bea.wlevs.management.Activate"

- Section G.3, "com.bea.wlevs.management.Prepare"

Another strategy is to always call the getDataSource() prior to using the data source. That is, the application code does not store a reference to the data source as a field in the component.

1.6.3 Dynamic Resource Lookup Using JNDI

Oracle CEP supports the use of JNDI to look up resources dynamically as Example 1–28.

Example 1–28   Dynamic Resource Lookup Using JNDI

```
import javax.naming.InitialContext;

public class SimpleBean implements EventBean {
...

    public abstract void getDataSource () throws Exception {
        InitialContext initialContext= new InitialContext ();
        return initialContext.lookup ("StockDs");
    }
}
```

In Example 1–28, the JNDI name StockDs must exactly match the name of the data source in the Oracle CEP server config.xml file as Example 1–19 shows.

> **Note:** You must disable security when starting the Oracle CEP server in order to use JNDI. Oracle does not recommend the use of JNDI for this reason.
>
> For more information, see "Configuring Security for Oracle CEP" in the *Oracle CEP Administrator's Guide*.

1.6.4 Understanding Resource Name Resolution

Oracle CEP server resolves resource names by examining the naming scopes that Table 1–3 lists.

Table 1–3   Resource Name Resolution

Naming Scope	Contents	Resolution Behavior
Component	The property names of the component's custom configuration	Mapping
Application	The names of the configuration elements in the application configuration files	Matching
Server	The names of the configuration elements in the server configuration file	Matching
JNDI	The names registered in the server's JNDI registry	Matching

Each naming scope contains a set of unique names. The name resolution behavior is specific to a naming scope. Some naming scopes resolve names by simple matching. Other scopes resolve names by mapping the name used to do the lookup into a new name. Once a name is mapped, lookup proceeds recursively beginning with the current scope.

1.7 Next Steps

After you have programmed all components of your application and created their configuration XML files:

- Assemble all the components into a deployable OSGi bundle. This step also includes creating the `MANIFEST.MF` file that describes the bundle.

  See Section 14.2, "Assembling an Oracle CEP Application."

- Start Oracle CEP.

  See:

  - Section 4.3.1, "How to Start an Oracle CEP Server"

  - Section 4.3.2, "How to Stop an Oracle CEP Server"

- Optionally configure the server in your domain to enable logging, debugging, and other services.

  See:

  - Section 4.2, "Creating Oracle CEP Servers"

  - Section 4.3, "Managing Oracle CEP Servers"

  - "Understanding Oracle CEP Configuration" in the *Oracle CEP Administrator's Guide*

- Deploy the application to Oracle CEP.

  See Section 14.3, "Deploying Oracle CEP Applications."

- Optionally, use the Oracle CEP load generator and Oracle CEP Visualizer to test and tune your Oracle CEP application.

  The Oracle CEP load generator is a testing tool that you can use to test your application, in particular its rules. This testing tool can temporarily replace the adapter component in your application, for testing purposes. For details, see Chapter 15, "Testing Applications With the Load Generator and csvgen Adapter."

  The Oracle CEP Visualizer is a runtime administration console with advanced diagnostic and maintenance features that you can use to manage and tune your Oracle CEP application. For details, see *Oracle CEP Visualizer User's Guide*.d

  See also Section 4.4, "Debugging an Oracle CEP Application Running on an Oracle CEP Server".

Part II

Oracle CEP IDE for Eclipse

Part II contains the following chapters:

2

Overview of the Oracle CEP IDE for Eclipse

Oracle CEP IDE for Eclipse is an IDE targeted specifically to programmers that want to develop Oracle CEP applications.

This chapter describes:

2.1 Overview of Oracle CEP IDE for Eclipse

Oracle CEP IDE for Eclipse is a set of plugins for the Eclipse IDE designed to help develop, deploy, and debug applications for Oracle CEP.

The key features of the Oracle CEP IDE for Eclipse are as follows:

- Project creation wizards and templates to quickly get started building event driven applications.
- Advanced editors for source files including Java and XML files common to Oracle CEP applications.
- Integrated server management to seamlessly start, stop, and deploy to Oracle CEP server instances all from within the IDE.
- Integrated debugging.
- Event Processing Network (EPN) visual design views for orienting and navigating in event processing applications and visually creating and editing EPN components.
- Oracle CEP application source file validation including Oracle Continuous Query Language (Oracle CQL) syntax highlighting and component configuration and assembly files.
- Ability to build and export deployable Oracle CEP applications.
- Integrated support for the Oracle CEP Visualizer so you can use the Oracle CEP Visualizer from within the IDE.

In Release 11gR1 (11.1.1), Oracle CEP IDE for Eclipse requires JDK 6.0. For more information, see:

- "Setting Your Development Environment" in the *Oracle CEP Getting Started*

- Section 2.4, "Configuring Eclipse"

2.2 Installing the Latest Oracle CEP IDE for Eclipse

New versions of the IDE will be made available via the Oracle Technology Network Web site. Oracle recommends that you install the IDE from this Eclipse update site:

http://download.oracle.com/technology/software/cep-ide/11/

To install the latest Oracle CEP IDE for Eclipse:

1. Obtain the required versions of Eclipse (3.3.2) and WTP (2.0). We recommend you take the entire Europa installation available at the following Web sites:

   Windows:
   http://www.eclipse.org/downloads/download.php?file=/technology/epp/downloads/release/europa/winter/eclipse-jee-europa-winter-win32.zip

   Linux:
   http://www.eclipse.org/downloads/download.php?file=/technology/epp/downloads/release/europa/winter/eclipse-jee-europa-winter-linux-gtk.tar.gz

   > **Note:** Check the Oracle CEP Web site (http://www.oracle.com/technologies/soa/complex-event-processing.html) for the latest requirements on the Eclipse tools.

2. Open your Eclipse IDE and select the menu item **Help > Software Updates > Find and Install**.

   The Install/Update dialog appears as Figure 2–1 shows.

Figure 2–1   Install/Update Dialog
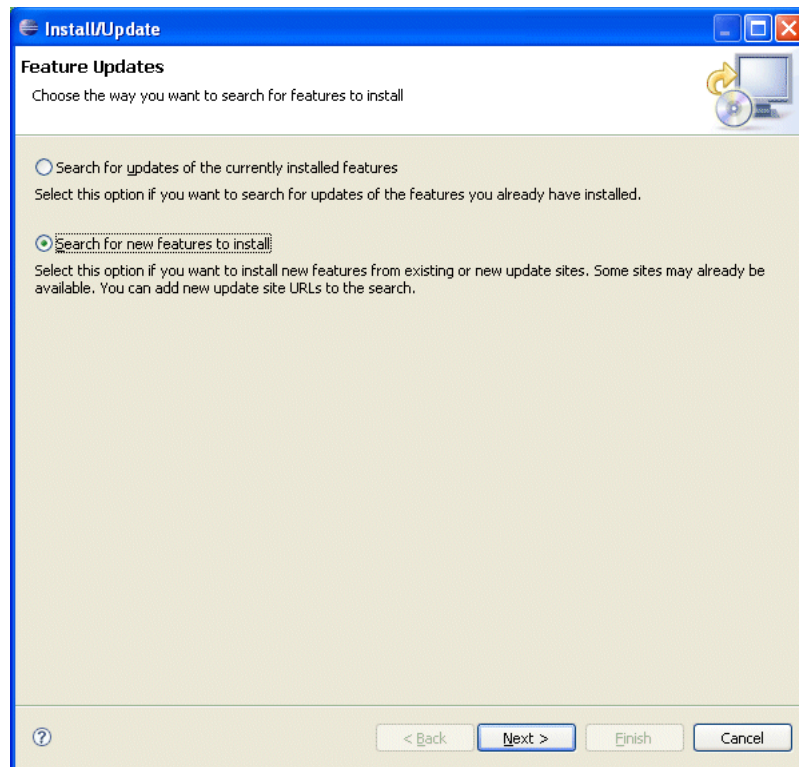
3. Select the **Search for New Features** option.

4. Click **Next**.

The Update Sites to Visit dialog appears as Figure 2–2 shows.

Figure 2–2 Update Sites to Visit Dialog
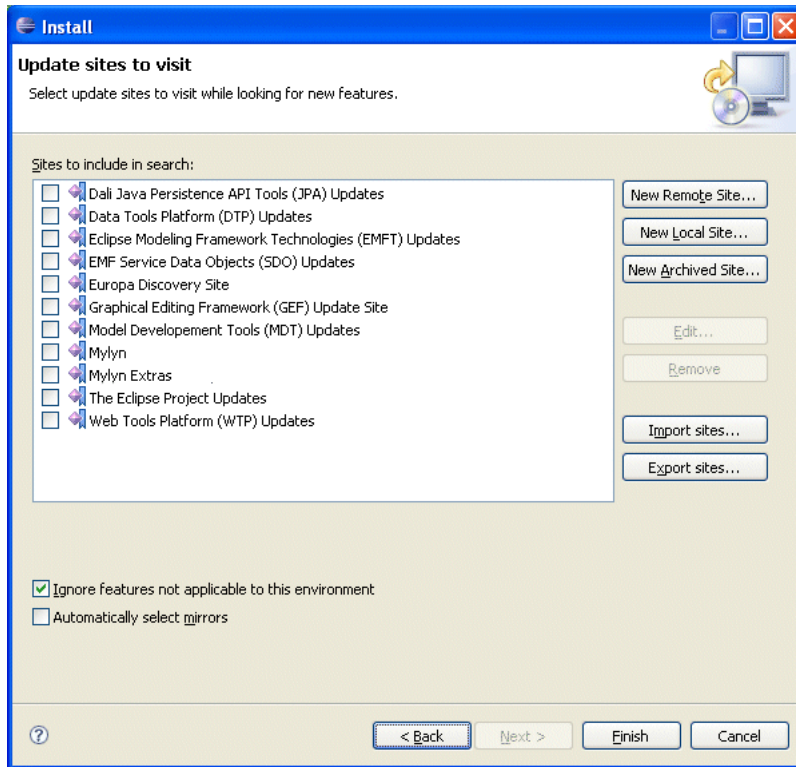
5. Click **New Remote Site**.

The New Update Site dialog appears as Figure 2–3 shows.

Figure 2–3   New Update Site Dialog
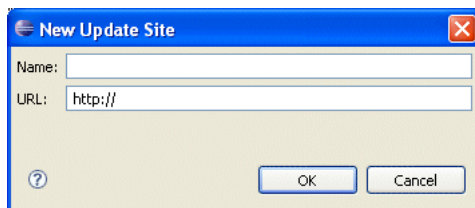
6. Configure this dialog as Table 2–1 describes.

Table 2–1   New Update Site Dialog Attributes

Attribute	Description
Name	The name for this remote update site. For example: `Oracle CEP Tools Update`.
URL	The URL to the Oracle Technology Network Web site. Valid value: `http://download.oracle.com/technology/software/cep-ide/11/`

7. Click **OK**.

8. Complete the Update Manager, selecting to install the Oracle CEP tools.

9. When prompted restart, Eclipse. If you skip this, unreliable behavior can occur.

10. To confirm the installation, select **Help > About Eclipse Platform**.

The About Eclipse Platform dialog appears as Figure 2–4 shows.

Figure 2–4   About Eclipse Platform

11. Click **Plug-In Details**.

The About Eclipse Platform Plug-ins dialog appears as Figure 2–5 shows.

Figure 2–5   About Eclipse Platform Plug-ins Dialog
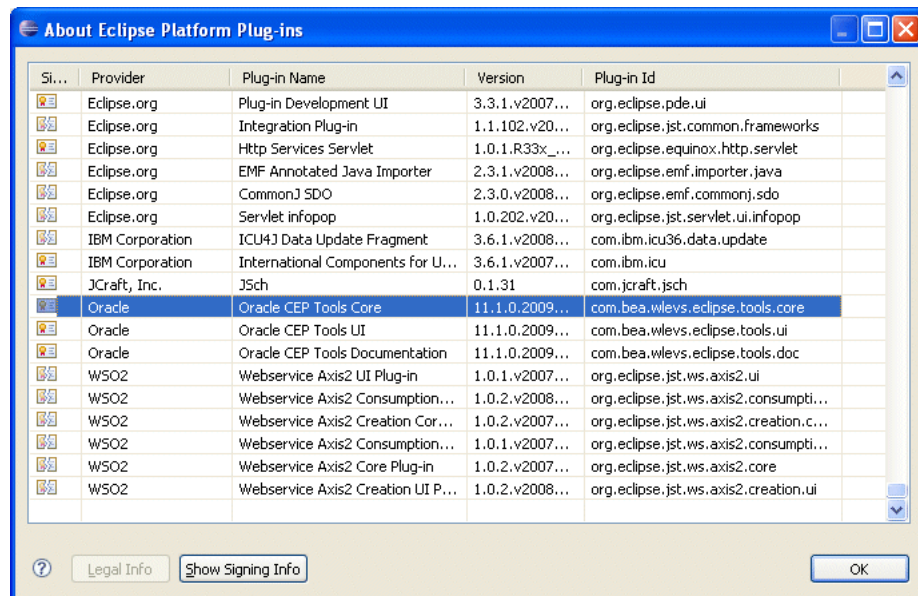
12. Scroll down in the list and confirm that the plug-ins that Table 2–2 lists are shown.

Table 2–2   Oracle CEP IDE for Eclipse Plug-Ins

Provider	Plug-in Name	Plug-in Id
Oracle	Oracle CEP Tools Core	`com.bea.wlevs.eclipse.tools.core`
Oracle	Oracle CEP Tools UI	`com.bea.wlevs.eclipse.tools.ui`
Oracle	Oracle CEP Tools Documentation	`com.bea.wlevs.eclipse.tools.doc`

13. After installing Oracle CEP IDE for Eclipse, consider the following configuration topics:

- Section 2.4, "Configuring Eclipse"

2.3 Installing the Oracle CEP IDE for Eclipse Distributed With Oracle CEP

A version of the Oracle CEP IDE for Eclipse is shipped with the Oracle CEP product, although this version might be older than the one on the Oracle Technology Network site.

To install the Oracle CEP IDE for Eclipse distributed with Oracle CEP:

1. Obtain the required versions of Eclipse (3.3.2) and WTP (2.0). We recommend you take the entire Europa installation available at the following Web sites:

Windows:

```
http://www.eclipse.org/downloads/download.php?file=/technolog
y/epp/downloads/release/europa/winter/eclipse-jee-europa-wint
er-win32.zip
```

Linux:

```
http://www.eclipse.org/downloads/download.php?file=/technolog
y/epp/downloads/release/europa/winter/eclipse-jee-europa-wint
er-linux-gtk.tar.gz
```

2. Open your Eclipse IDE and select the menu item **Help > Software Updates > Find and Install**.

The Install/Update dialog appears as Figure 2–1 shows.

Figure 2–6   Install/Update Dialog
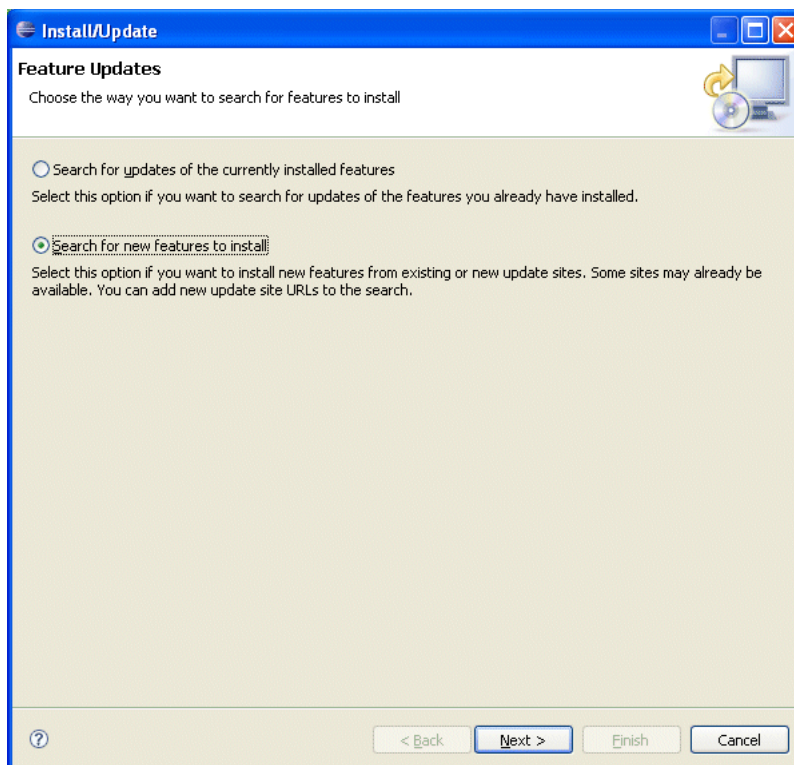
3. Select the **Search for New Features** option.

4. Click **Next**.

The Update Sites to Visit dialog appears as Figure 2–2 shows.

Figure 2–7   Update Sites to Visit Dialog
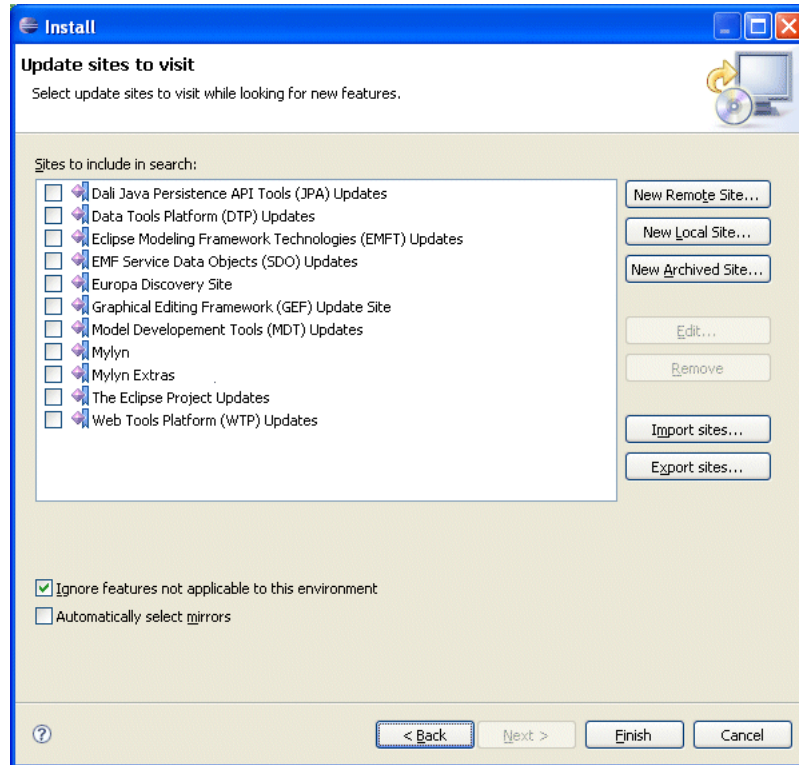
5. Click **New Archived Site**.

The Select Local Site Archive dialog appears as Figure 2–8 shows.

Figure 2–8   Select Local Site Archive Dialog
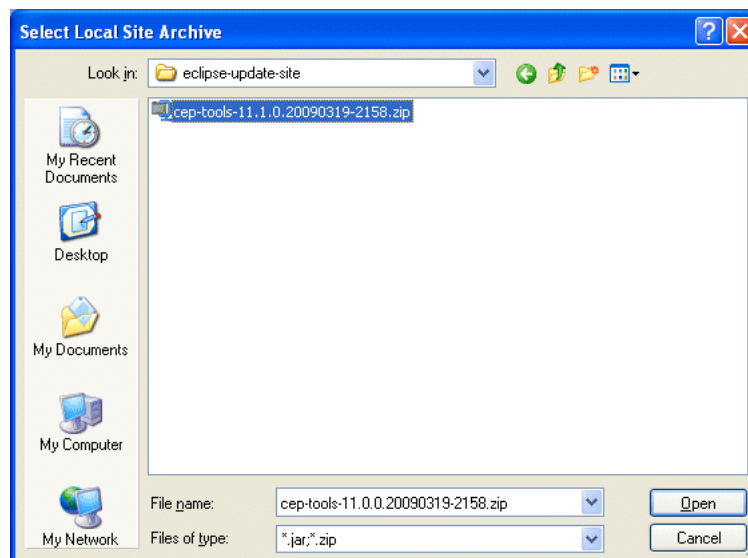
6. Navigate to the *ORACLE_CEP_HOME*/ocep_11.1/eclipse-update-site directory and select the cep-tools-11.0.0.*DATE-BUILD*.zip file.

   Where *ORACLE_CEP_HOME* refers to the main directory into which you installed Oracle CEP, such as /oracle_cep, and *DATE* is the build date and *BUILD* is the build number.

7. Click **Open**.

8. Complete the Update Manager, selecting to install the Oracle CEP tools.

9. When prompted restart, Eclipse. If you skip this, unreliable behavior can occur.

10. To confirm the installation, select **Help > About Eclipse Platform**.

    The About Eclipse Platform dialog appears as Figure 2–4 shows.

Figure 2–9   About Eclipse Platform

11. Click **Plug-In Details**.

    The About Eclipse Platform Plug-ins dialog appears as Figure 2–5 shows.

Figure 2–10   About Eclipse Platform Plug-ins Dialog
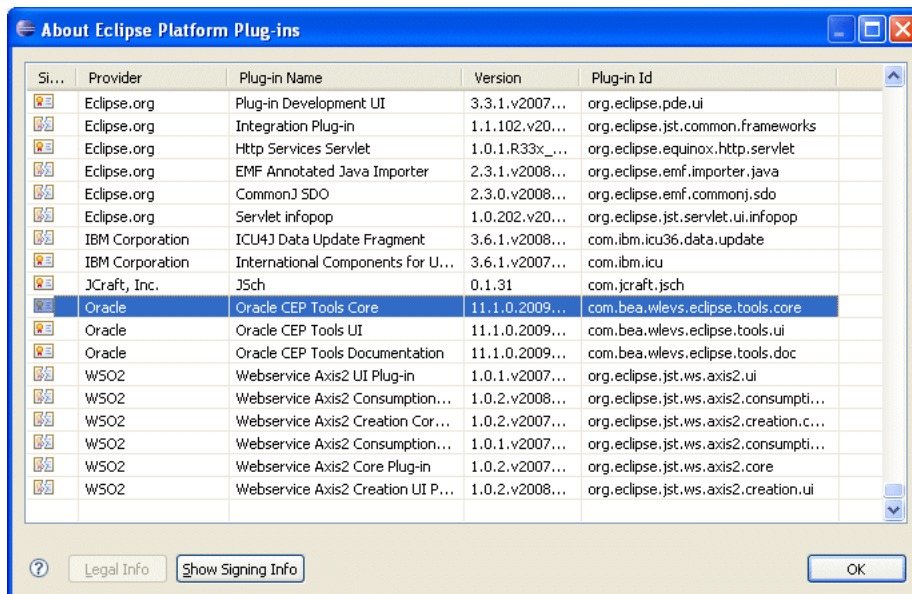
12. Scroll down in the list and confirm that the plug-ins that Table 2–2 lists are shown.

Table 2–3   Oracle CEP IDE for Eclipse Plug-Ins

Provider	Plug-in Name	Plug-in Id
Oracle	Oracle CEP Tools Core	`com.bea.wlevs.eclipse.tools.core`
Oracle	Oracle CEP Tools UI	`com.bea.wlevs.eclipse.tools.ui`
Oracle	Oracle CEP Tools Documentation	`com.bea.wlevs.eclipse.tools.doc`

13. After installing Oracle CEP IDE for Eclipse, consider the following configuration topics:

- Section 2.4, "Configuring Eclipse"

2.4 Configuring Eclipse

This section describes how to configure Eclipse to work with the Oracle CEP.

To configure Eclipse:

1. Exit out of Eclipse if it is running.

2. Install a Java 6 JRE on your computer.

The JRockit Java 6.0 JRE is installed with Oracle CEP in your Oracle CEP home directory. For example:

```
C:\OracleCEP\jrockit-R27.6.0-50-1.6.0_05\jre
```

3. Using the editor of your choice, open your `eclipse.ini` file located in your Eclipse install directory, for example, `C:\eclipse` as Example 2–1 shows.

Example 2–1   Default eclipse.ini File

```
-showsplash
org.eclipse.platform
--launcher.XXMaxPermSize
256M
```

> **Note:** When making changes to the `eclipse.ini` file, add arguments one argument per line, as http://wiki.eclipse.org/Eclipse.ini describes.
>
> For more information about configuring Eclipse, see http://wiki.eclipse.org/FAQ_How_do_I_run_Eclipse.

4. Add the following lines to the `eclipse.ini` file as Example 2–2 shows.

Example 2–2   Memory Resources

```
-showsplash
org.eclipse.platform
--launcher.XXMaxPermSize
256M
-vmargs
-Xmx512m
```

```
-XX:MaxPermSize=256M
```

5. Add the following to the `eclipse.ini` file as Example 2–3 shows.

Example 2–1 Virtual Machine Path

```
-vm
PATH-TO-JRE-6.0-JAVAW
```

Where `PATH-TO-JRE-6.0-JAVAW` is the fully qualified path to your Java 6.0 JRE `javaw` executable. For example:

```
-vm
C:\OracleCEP\jrockit-R27.6.0-50-1.6.0_05\jre\bin\javaw.exe
```

> **Note:** Do not put both the `-vm` and the path on the same line. Each must be on a separate line as Example 2–3 shows.

6. Save and close the `eclipse.ini` file.

7. Start Eclipse.

8. Select **Help > About Eclipse Platform** and click Configuration Details.

   The Configuration Details dialog appears as shown in Figure 2–11.

Figure 2–11 Configuration Details for Java 6
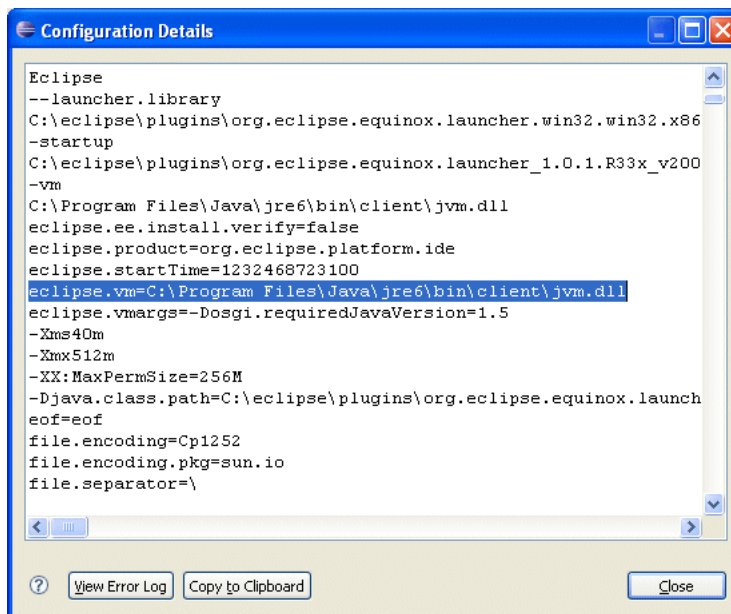
9. Confirm that the `eclipse.vm` property points to the Java 6.0 JRE you configured in the `eclipse.ini` file.

10. Open the Oracle CEP project you originally created using Java 5.

11. Select **Window > Preferences**.

    The Preferences dialog appears as shown in Figure 2–12.

Figure 2–12 Preferences Dialog
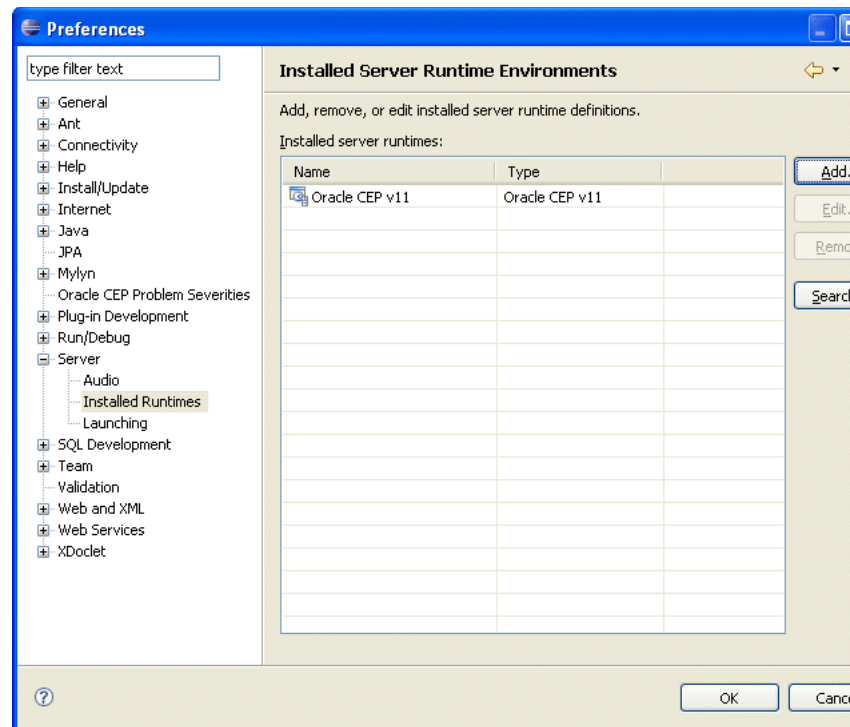
12. Expand the **Server** option and select **Installed Runtimes**.

13. Select a runtime that uses the Java 6 JRE.

14. Click **OK**.

15. Right-click your project and select **Properties**.

The Project Properties dialog appears as shown in Figure 2–13.

Figure 2–13   Project Properties Dialog: Java Build Path
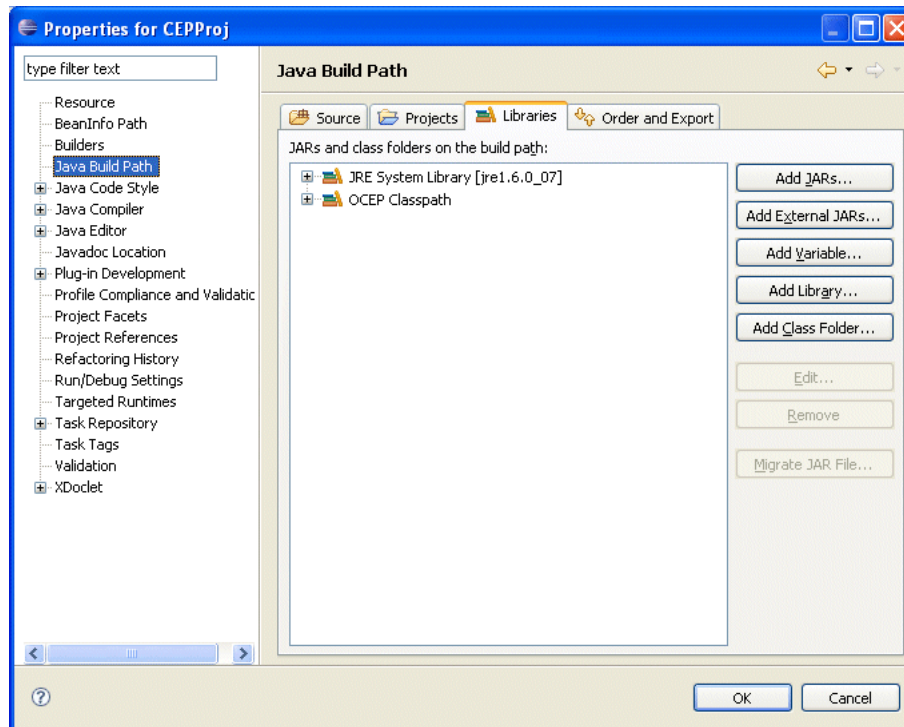
16. Select the **Java Build Path** option and click on the **Libraries** tab.

17. Select the existing **JRE System Library [jre1.5.0_XX]** Java 5.0 library and click **Remove**.

18. Create a new JRE System Library for Java 6.0 by clicking **Add Library** and choosing the Java 6.0 JRE.

19. Select the **Project Facets** option.

    The Project Facet properties are displayed as Figure 2–14 shows.

Figure 2–14 Project Properties Dialog: Project Facets
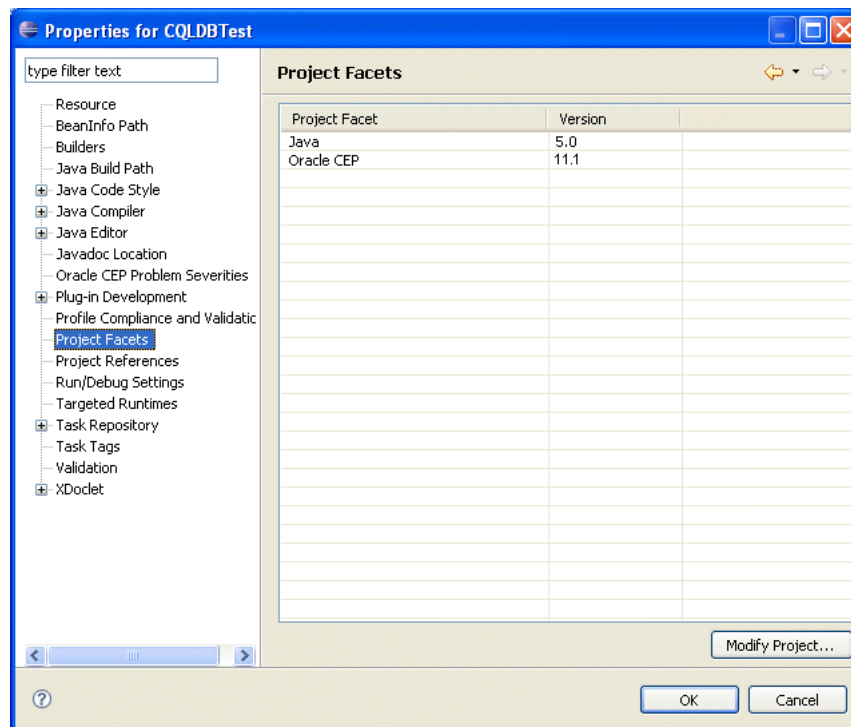
20. Click **Modify Project**.

The Modify Faceted Project dialog appears shown in Figure 2–15.

Figure 2–15 Modify Faceted Project
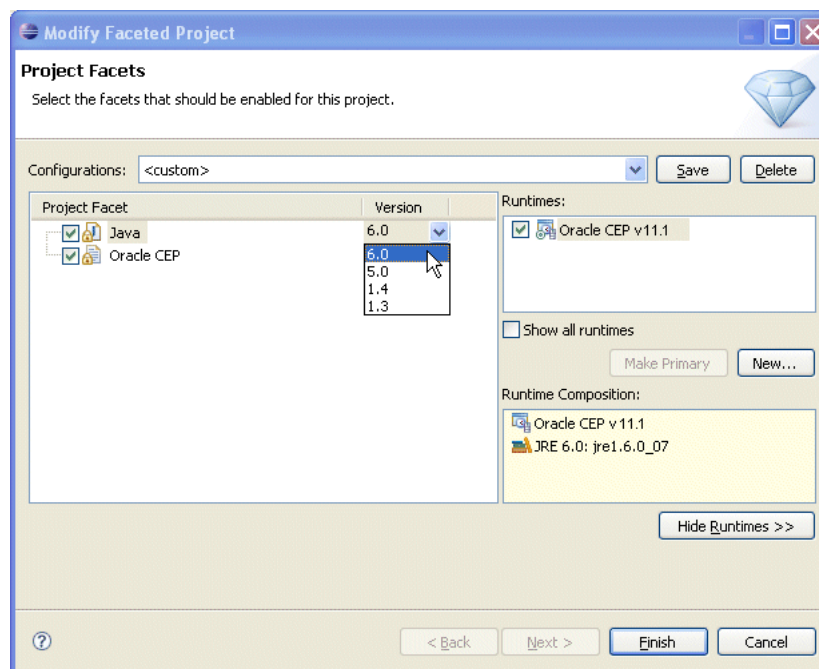
21. For the Java facet, select 6.0 from the **Version** pull-down menu.

22. Click **Finish**.

23. Click **OK**.

Your project will now build and run using Java 6.0.

3

Oracle CEP IDE for Eclipse Projects

An Oracle CEP project is an Eclipse project that brings together all Oracle CEP artifacts.

This chapter describes:

- Section 3.1, "Oracle CEP Project Overview"
- Section 3.2, "Creating Oracle CEP Projects"
- Section 3.3, "Exporting Oracle CEP Projects"
- Section 3.4, "Upgrading Projects"
- Section 3.5, "Managing Oracle CEP Project Libraries"
- Section 3.6, "Configuring Oracle CEP IDE for Eclipse Preferences"

3.1 Oracle CEP Project Overview

An Oracle CEP application includes the following artifacts:

- Java source files
- XML configuration files
- OSGi bundle Manifest file

Figure 3–1 shows the Explorer after creating a project.

Figure 3–1   Oracle CEP Project Structure
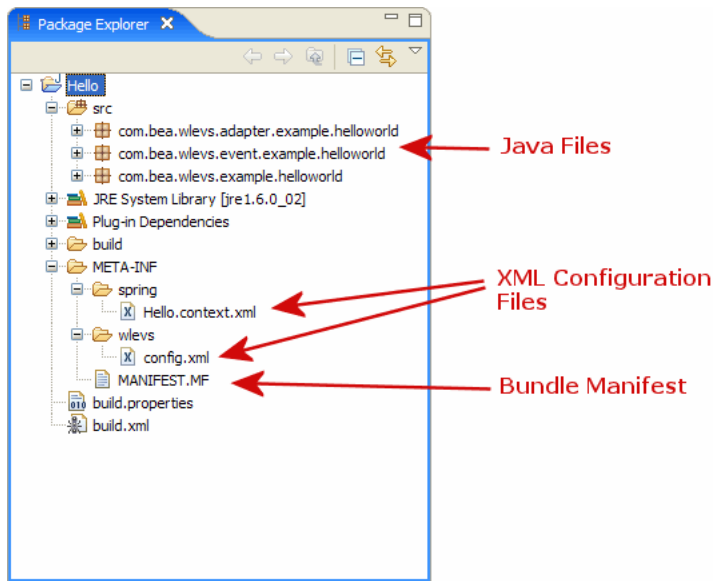
Table 3–1 summarizes the important files in an Oracle CEP project including their use and location.

Table 3–1   Oracle CEP Project Artifacts

File Type	Location	Description
Java source files	Any Java source folder. Default: `src`.	Events, adapters, and listeners are implemented in an Oracle CEP application with Java files. All Java files must be in a source folder in order to be compiled. For more information, see Chapter 1, "Overview of Creating Oracle CEP Applications".
EPN assembly file	`META-INF/spring`	These are the main files used to wire-up an EPN and to define event types. This is a Spring context file, and is where adapters, channels, processors, and listeners are connected. For more information, see Chapter 1.1.7, "EPN Assembly File".
Processor configuration file	`META-INF/wlevs`	The processor configuration file is where the Complex Event Processor (CEP) is defined. In this file you'll find processor rules (defined in the Continuous Query Language - CQL or the Event Processing Language--EPL) and other component configuration settings. For more information, see: ■ Chapter 10, "Configuring Oracle CQL Processors" ■ Chapter 11, "Configuring EPL Processors"
`MANIFEST.MF` file	`META-INF`	The manifest file contains metadata about your application including its name, version, and dependencies, among others. For more information, see Chapter 14, "Assembling and Deploying Oracle CEP Applications".

3.2 Creating Oracle CEP Projects

Development of an Oracle CEP application begins by creating a project to hold all source code and related files.

Projects correspond 1-to-1 with Oracle CEP applications and are the unit of work that is associated with and deployed to a server instance. In concrete terms, the output of a built project is a single OSGi bundle (JAR) containing the Oracle CEP application.

3.2.1 How to Create an Oracle CEP Project

By default new projects are set to use Java 6.0. This section describes how to create an Oracle CEP project using Java 6. For information on configuring an Oracle CEP project to use Java 6, see Section 2.4, "Configuring Eclipse".

To create an Oracle CEP project:

1. Open the EPN Editor (see Section 5.1, "Opening the EPN Editor")

2. Select **File > New Project**.

   The New Project - Select a Wizard dialog appears as shown in Figure 3–2.

Figure 3–2   New Project - Select a Wizard Dialog
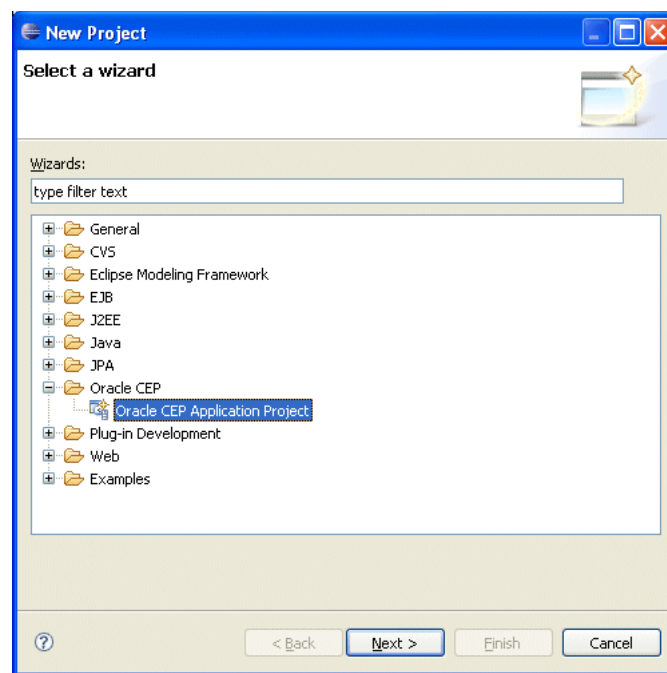
3. Expand **Oracle CEP** and select **Oracle CEP Application Project**.

4. Click **Next**.

   The New Oracle CEP Application Project wizard appears as shown in Figure 3–3.

Figure 3–3   New Oracle CEP Application Project Wizard: Create an Oracle CEP Application
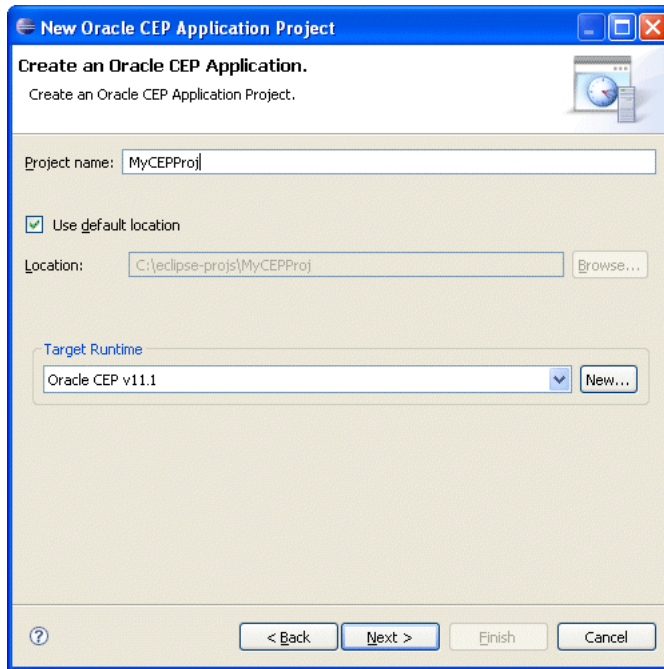
5. Configure the Create an Oracle CEP Application dialog as shown in Table 3–2.

Table 3–2   Create an Oracle CEP Application Dialog

Attribute	Description
Project name	The name of your Oracle CEP project. This name will be used as the default name of your application when it is deployed to the Oracle CEP server.
Location	The directory in which your project is stored. By default your project is located inside the Eclipse workspace directory. To keep your workspace and source code control directories separate, uncheck **Use default location** and click **Browse** to place the project in a directory outside of your workspace.
Target Runtime	The Oracle CEP server you will deploy your project to.

6. Click **Next**.

The Oracle CEP Application Content dialog appears as shown in Figure 3–4.

Figure 3–4 New Oracle CEP Application Project Wizard: Oracle CEP Application Content

7. Optionally, configure the Oracle CEP Application Content dialog as shown in Table 3–3.

Table 3–3 Oracle CEP Application Content Dialog

Attribute	Description
Bundle ID	The unique ID that distinguishes this application's OSGi bundle from those deployed to the target runtime.
Bundle Version	The version of this instance of this OSGi bundle.
Bundle Name	The name of this application's OSGi bundle.
Bundle Provider	The name of the provider for this application's OSGi bundle (optional).

8. Click **Next**.

The Template dialog appears as shown in Figure 3–5.

Figure 3–5   New Oracle CEP Application Project Wizard: Template Dialog
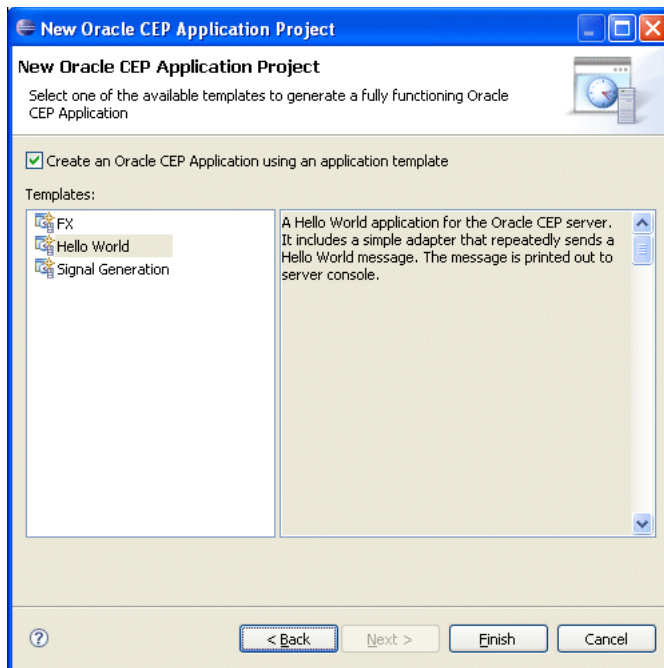
9. Optionally, select an Oracle CEP application template to pre-populate your project with the content that the template specifies.

10. Click **Finish**.

The Oracle CEP IDE for Eclipse creates the Oracle CEP project.

3.3 Exporting Oracle CEP Projects

Exporting an Oracle CEP project builds the project into an OSGi bundle that can be deployed to a production Oracle CEP server.

3.3.1 How to Export an Oracle CEP Project

This section describes how to export an Oracle CEP project into an OSGi bundle.

To export an Oracle CEP project:

1. Start the Oracle CEP IDE for Eclipse and open your Oracle CEP project.

2. The Oracle CEP IDE for Eclipse compiles and adds Java resources to the exported JAR automatically. If your project contains other resources (such as a manifest file or images), configure your project to export them:

   a. Locate the `build.properties` file in the Project Explorer and double-click this file to edit it.

   The `build.properties` file opens as shown in Figure 3–6.

Figure 3–6   Oracle CEP Project build.properties File
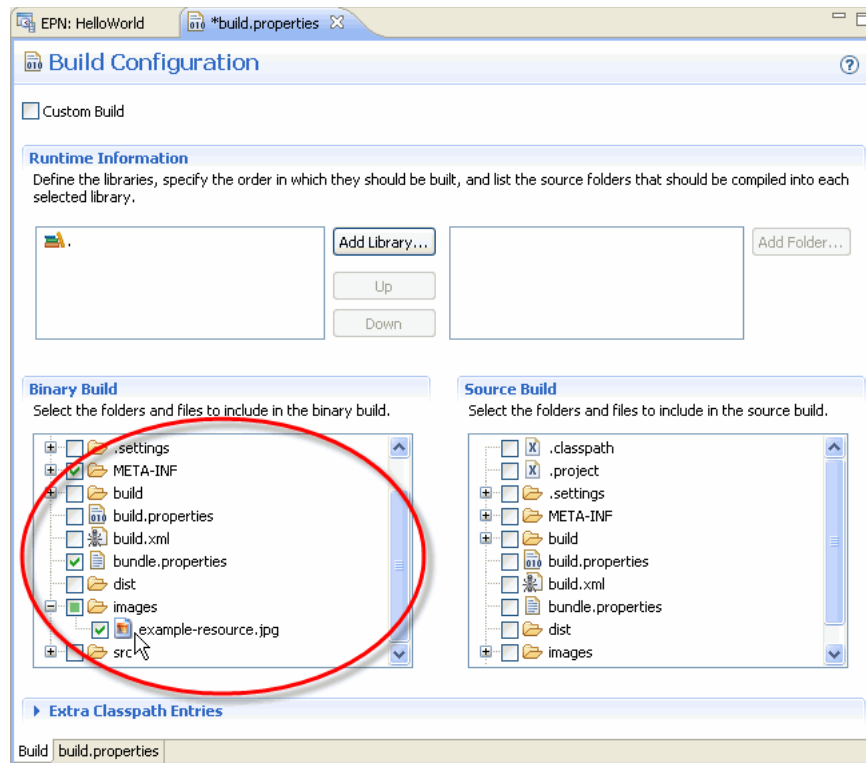
b. In the **Binary Build** area, check the resources you want exported with your application.

3. Select **File > Export**.

The Export dialog appears as shown in Figure 3–7.

Figure 3–7   Export Dialog
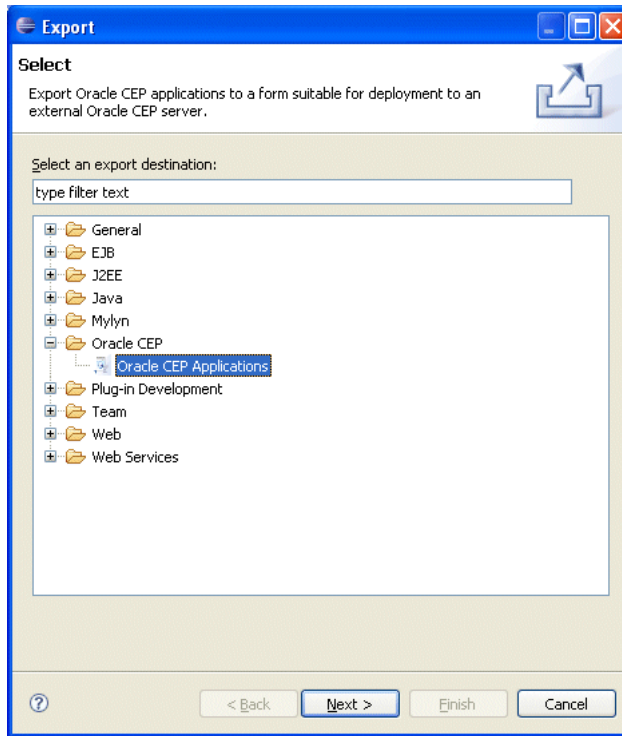
4. Expand the **Oracle CEP** option and select **Oracle CEP Applications**.

5. Click **Next**.

The Oracle CEP Applications Export: Select Project dialog appears as shown in Figure 3–8.

Figure 3–8   Oracle CEP Applications Export: Select Project Dialog
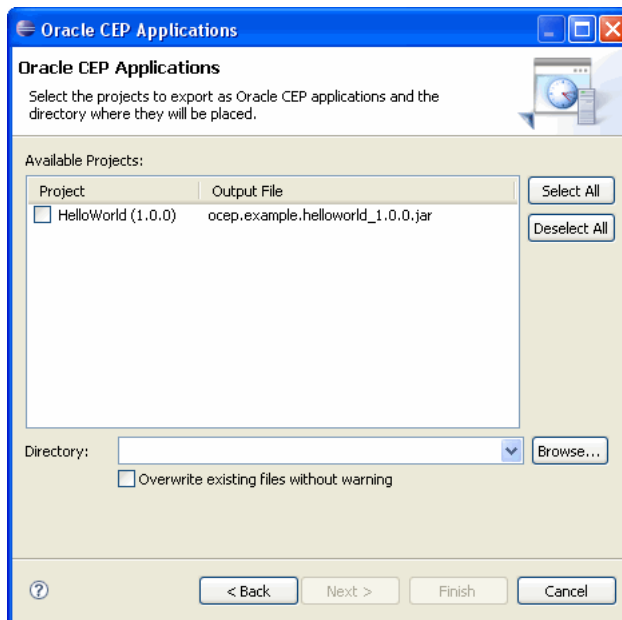
6. Configure the Oracle CEP Applications Export: Select Project dialog as shown in Table 3–4.

Table 3–4 Oracle CEP Application Content Dialog

Attribute	Description
Available Projects	The list of Oracle CEP projects available for export.
	Check the project or projects you want to export.
	Each project will be exported to a JAR file with the name given in the Output File column. The name of the bundle that will be exported conforms to the OSGi bundle naming conventions, using the bundle ID and the bundle version in the JAR name.
Directory	The directory in which Oracle CEP project JAR files are exported.
	Click **Browse** to choose this directory.
Overwrite existing files without warning	Check this option to overwrite existing JAR files with the same name in the selected directory.

7. Click **Finish**.

   Your project, its Java resources, and any binary resources you selected are exported to the project JAR file.

8. Deploy the JAR file to your Oracle CEP server.

   See Section 4.3.5, "How to Deploy an Application to an Oracle CEP Server".

9. Deploy other dependent resources, if any, to your Oracle CEP server.

   For example:

   - Other OSGi bundles your application depends on.

     Deploy these bundles on the Oracle CEP server using the Oracle CEP Visualizer or command line deployment tools.

   - Any entries in `config.xml` for datasources that are referenced from within the application.

     Add these entries to the target server's `config.xml` file.

3.4 Upgrading Projects

When upgrading Oracle CEP from one version to another, it may be necessary to make changes to your existing Oracle CEP projects.

This section describes:

- Section 3.4.1, "How to Upgrade Projects from Oracle CEP 2.1 to 10.3"

- Section 3.4.2, "How to Upgrade Projects from Oracle CEP 10.3 to Release 11gR1 (11.1.1)"

For more information, see:

- Section 2.4, "Configuring Eclipse"

- *Oracle CEP Getting Started*

3.4.1 How to Upgrade Projects from Oracle CEP 2.1 to 10.3

While project structure has stayed the same since 2.1, the data stored in Oracle CEP Projects has changed significantly. It is therefore necessary to take steps to upgrade 2.1 projects manually before continuing their development in 10.3.

The following outlines the steps necessary to upgrade 2.1 projects to 10.3

To upgrade projects from Oracle CEP 2.1 to 10.3:

1. Open your Oracle CEP 2.1 project in Oracle CEP IDE for Eclipse.

2. Select **File > Switch Workspace > Other**.

   The Workspace Launcher dialog appears as shown in Figure 3–9.

Figure 3–9   Workspace Launcher Dialog
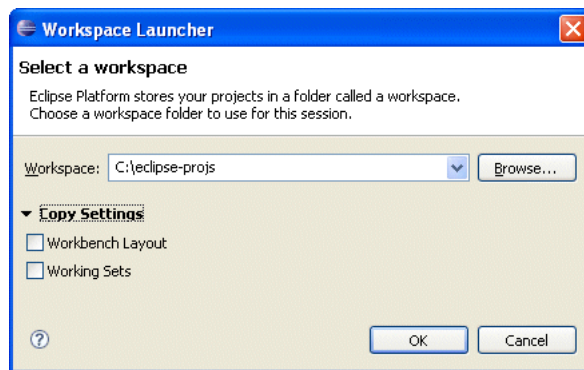
Note: Do not choose to copy settings from the current workspace.

---

3. Click Browse and select a new workspace directory.

   Eclipse exits and restarts using the new workspace.

4. Select **File > Import**.

   The Import Dialog appears as shown in Figure 3–10.

Figure 3–10 Import Dialog
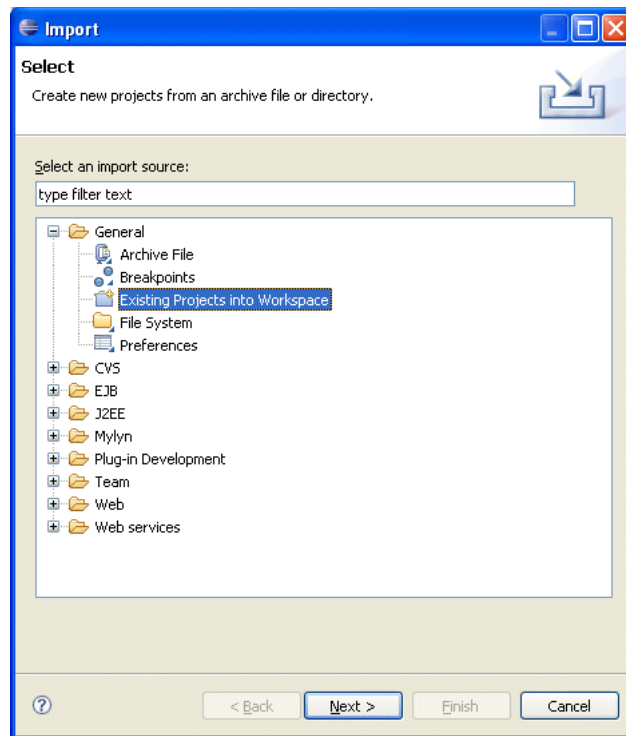
5. Expand the **General** option and select **Existing Projects into Workspace**.

6. Click **Next**.

    The Import Projects dialog appears as shown in Figure 3–11.

Figure 3–11   Import Projects Dialog
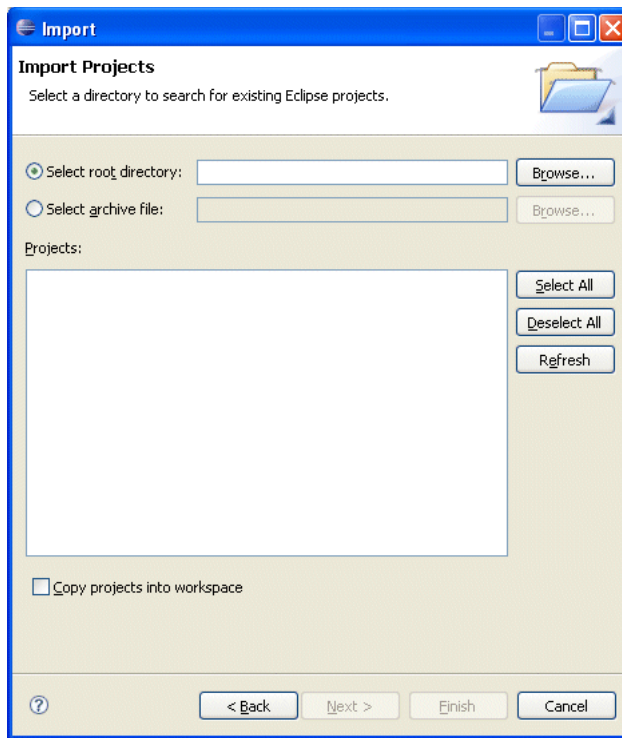
7. Use the Import Projects dialog to import your 2.1 projects into the new workspace. Optionally, choose to copy the project files into your new workspace.

8. For each project, change the project facet version as follows:

   a. Right-click your project and select **Properties**.

      The Project Properties dialog appears as shown in Figure 3–12.

Figure 3–12   Project Properties Dialog: Project Facets
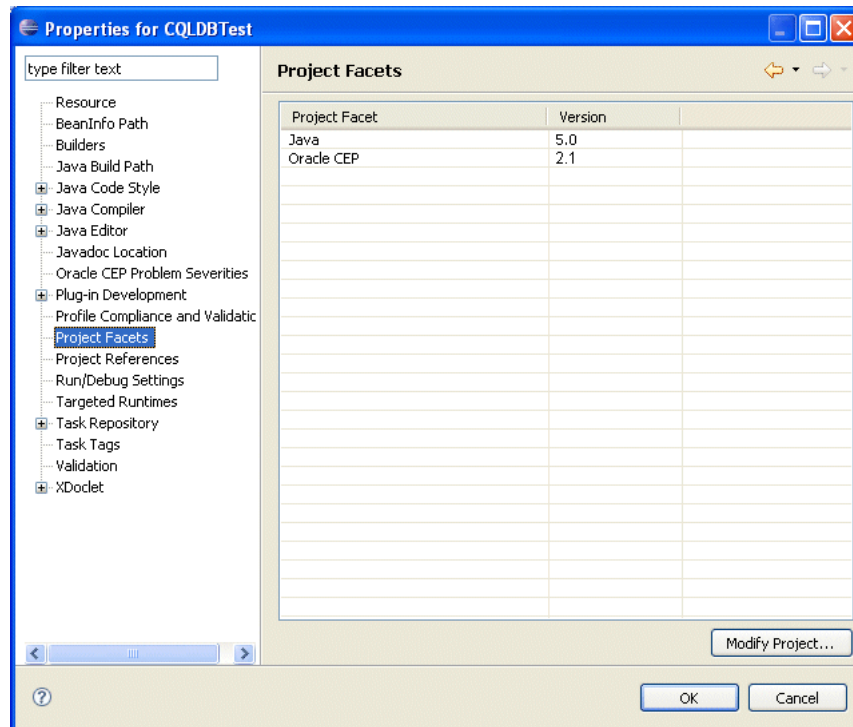
b. Select the Project Facets option.

The Project Facet properties are displayed as Figure 3–12 shows.

c. Click **Modify Project**.

The Modify Faceted Project dialog appears shown in Figure 3–13.

Figure 3–13   Modify Faceted Project
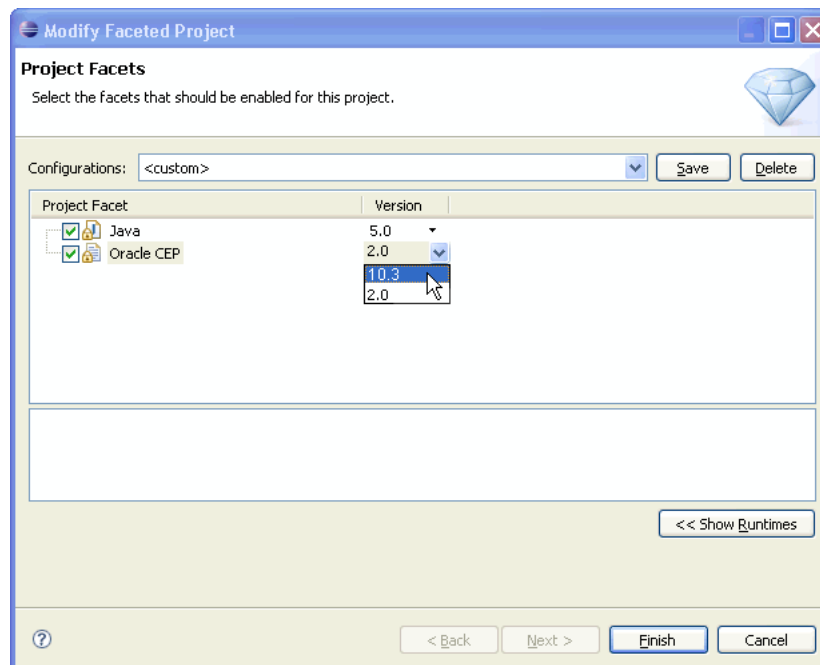
    d. For the Oracle CEP facet, select 10.3 from the **Version** pull-down menu.

    e. Click **Finish**.

    f. Click **OK**.

    g. Repeat for the next project.

9. Create a new Oracle CEP server runtime:

    a. Select **Window > Preferences**.

    The Preferences dialog appears as shown in Figure 3–14.

Figure 3–14   Preferences Dialog
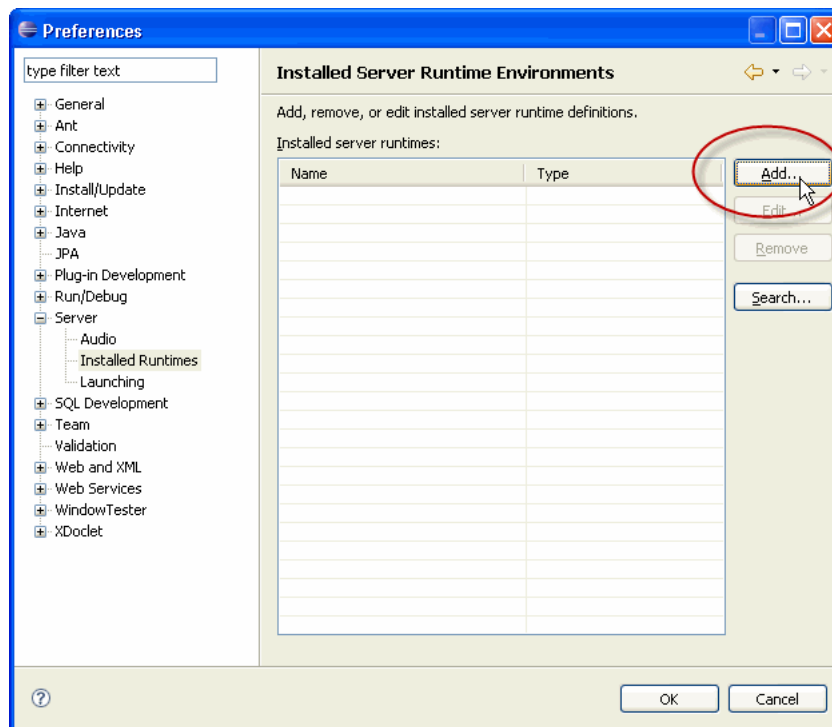
    b. Expand the **Server** option and select **Installed Runtimes**.

    c. Add a new 10.3 Oracle CEP server runtime as Section 4.2.1, "How to Create an Oracle CEP Server Runtime" describes.

    d. Click **OK**.

10. For each project, specify the new 10.3 Oracle CEP server runtime you created:

    a. Right-click your project and select **Properties**.

    The Project Properties dialog appears as shown in Figure 3–15.

Figure 3–15   Project Properties Dialog: Targeted Runtimes
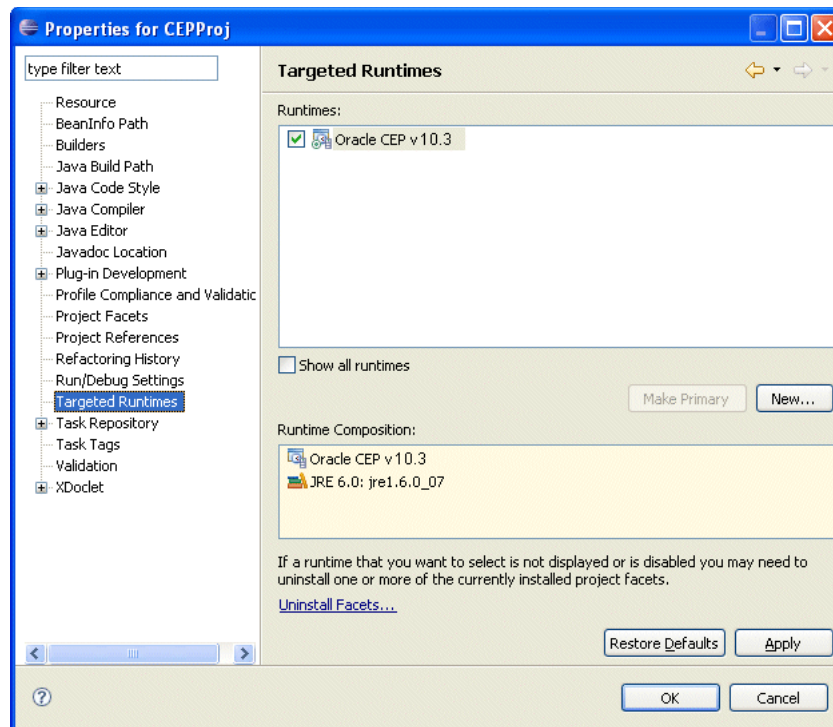
b. Select the **Targeted Runtimes** option.

The Targeted Runtimes properties are displayed as Figure 3–15 shows.

c. Check the new Oracle CEP 10.3 targeted runtime you created.

d. Click **OK**.

e. Repeat for the next project.

11. For each project update the project builders:

a. Right-click the project and select **Open EPN Editor**.

b. If the EPN diagram opens without error, proceed to step 12.

c. If the EPN diagram opens with the error shown in Figure 3–16, click the **Configure Builder** button.

Figure 3–16   Builder Error
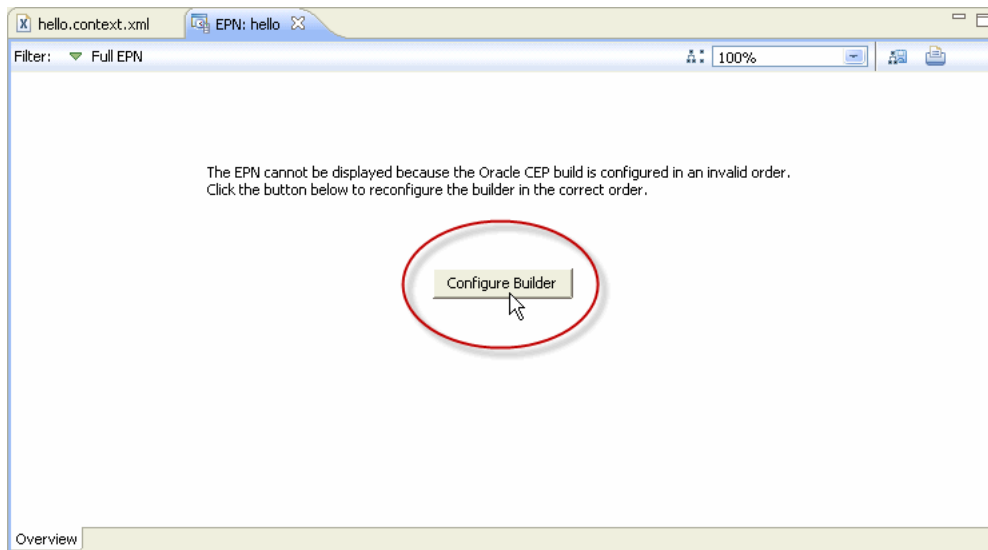
> The EPN diagram is displayed.

> **d.** Repeat for the next project.

12. Validate build inclusions:

> If your application bundle is using bundle localization and has substitution variables in its `MANIFEST.MF` file such as:

```
Bundle-Name: %project.name
```

> Then your project root directory's `build.properties` file element `bin.include` must contain a reference to your `bundle.properties` file such as:

```
bin.includes = META-INF/,\
               bundle.properties,\
               .
```

13. Perform source changes, if necessary.

> For more information, see:

> - "Upgrading a WebLogic Event Server 2.0 Application to Run on Oracle CEP 10.3" in the *Oracle CEP Getting Started*

> - *Oracle CEP Release Notes* for 10.3 (http://download.oracle.com/docs/cd/E13157_01/wlevs/docs30/notes/notes.html)

> After performing these steps, you should do a clean build of your project.

3.4.2 How to Upgrade Projects from Oracle CEP 10.3 to Release 11*g*R1 (11.1.1)

> While project structure has stayed the same since 10.3, the data stored in Oracle CEP Projects has changed significantly. It is therefore necessary to take steps to upgrade 10.3 projects manually before continuing their development in Release 11*g*R1 (11.1.1).

> The following outlines the steps necessary to upgrade 10.3 projects to Release 11*g*R1 (11.1.1)

To upgrade projects from Oracle CEP 10.3 to Release 11*g*R1 (11.1.1)

1. Open your Oracle CEP 10.3 project in Oracle CEP IDE for Eclipse.

2. Select **File > Switch Workspace > Other**.

   The Workspace Launcher dialog apperas as shown in Figure 3–17.

Figure 3–17   Workspace Launcher Dialog
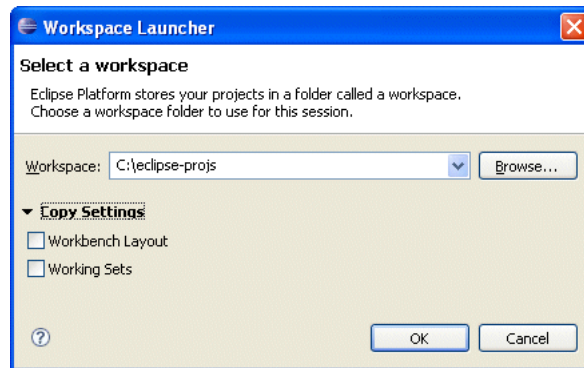
3. Click Browse and select a new workspace directory.

> **Note:**   Do not choose to copy settings from the current workspace.

   Eclipse exits and restarts using the new workspace.

4. Select **File > Import**.

   The Import Dialog appears as shown in Figure 3–10.

Figure 3–18   Import Dialog
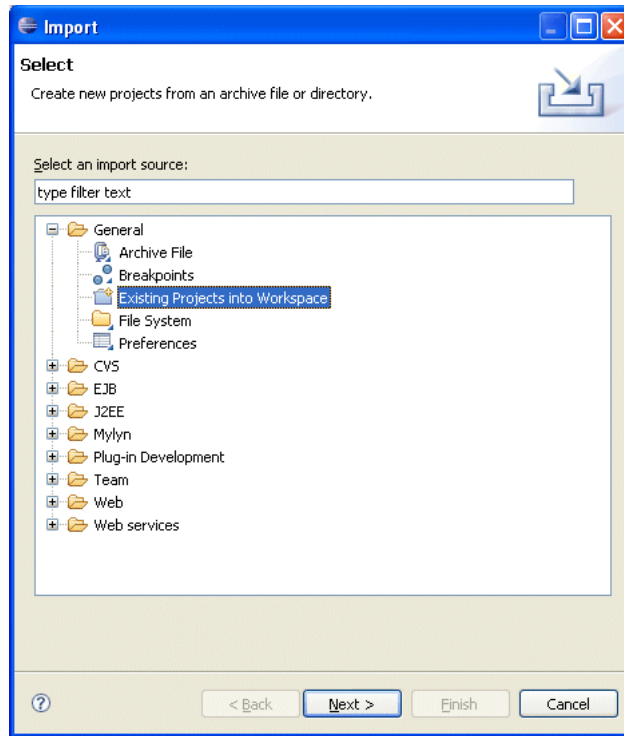
5. Expand the **General** option and select **Existing Projects into Workspace**.

6. Click **Next**.

    The Import Projects dialog appears as shown in Figure 3–11.

Figure 3–19   Import Projects Dialog
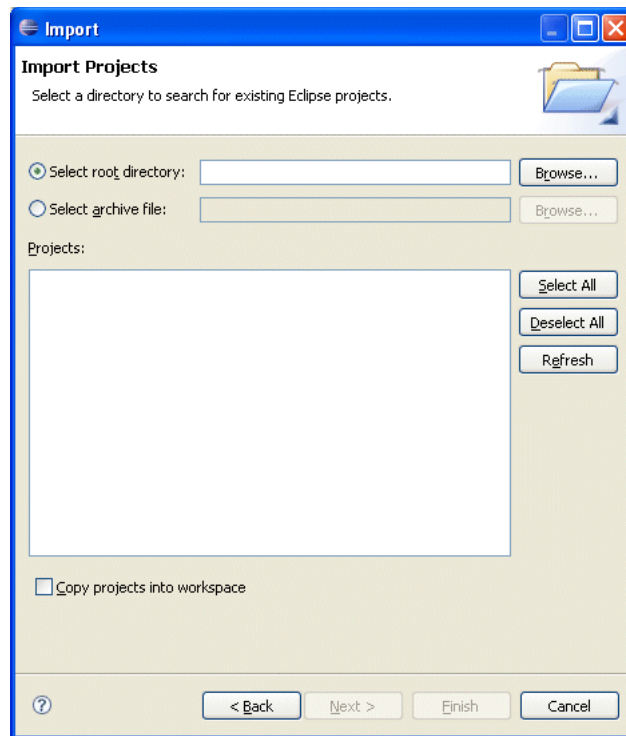
7. Use the Import Projects dialog to import your 10.3 projects into the new workspace. Optionally, choose to copy the project files into your new workspace.

8. For each project, change the project facet version as follows:

   a. Right-click your project and select **Properties**.

   The Project Properties dialog appears as shown in Figure 3–12.

Figure 3–20 Project Properties Dialog: Project Facets
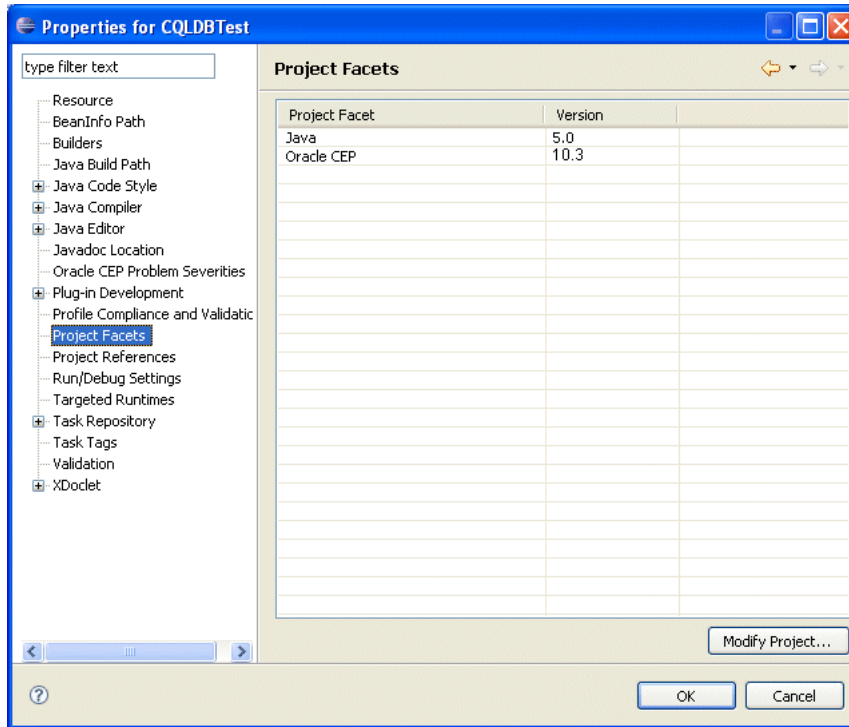
b. Select the Project Facets option.

The Project Facet properties are displayed as Figure 3–12 shows.

c. Click **Modify Project**.

The Modify Faceted Project dialog appears shown in Figure 3–13.

Figure 3–21 Modify Faceted Project
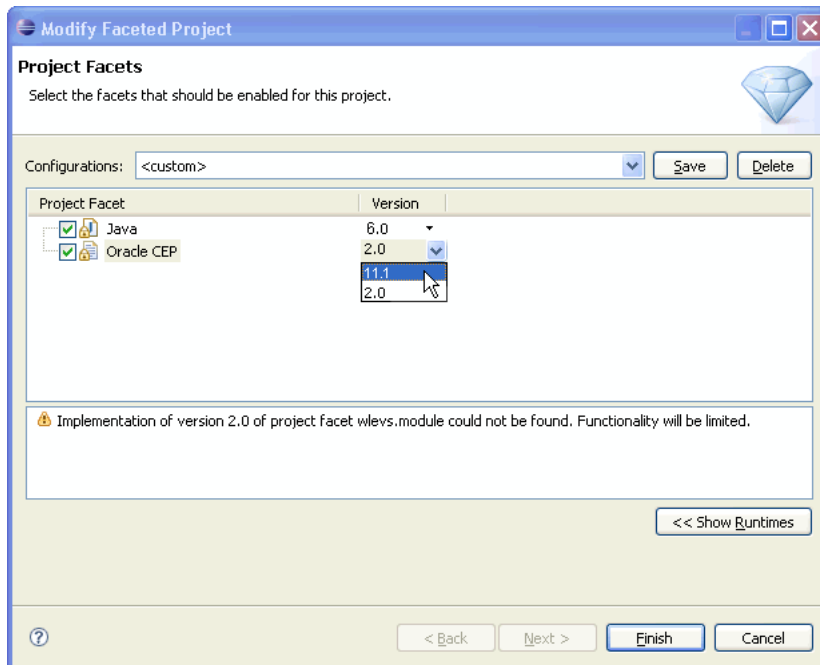
    d. For the Java facet, select 6.0 from the **Version** pull-down menu.

    e. For the Oracle CEP facet, select 11.1 from the **Version** pull-down menu.

    f. Click **Finish**.

    g. Click **OK**.

    h. Repeat for the next project.

9. Create a new Oracle CEP server runtime:

    a. Select **Window > Preferences**.

    The Prefences dialog appears as shown in Figure 3–14.

Figure 3–22   Preferences Dialog
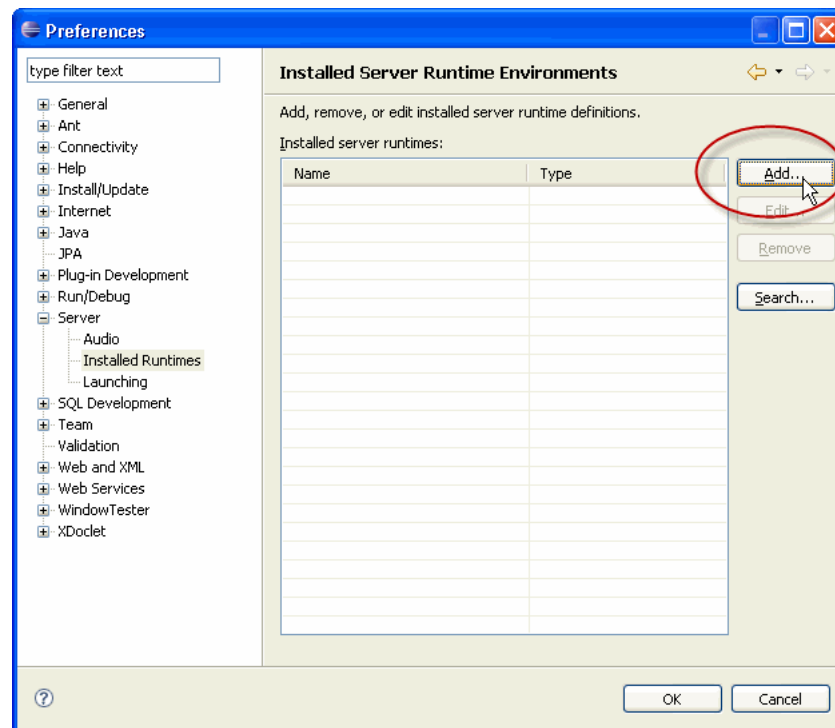
    b. Expand the **Server** option and select **Installed Runtimes**.

    c. Add a new 11.0 Oracle CEP server runtime as Section 4.2.1, "How to Create an Oracle CEP Server Runtime" describes.

    d. Click **OK**.

10. For each project, specify the new 11.0 Oracle CEP server runtime you created:

    a. Right-click your project and select **Properties**.

    The Project Properties dialog appears as shown in Figure 3–15.

Figure 3–23 Project Properties Dialog: Targeted Runtimes
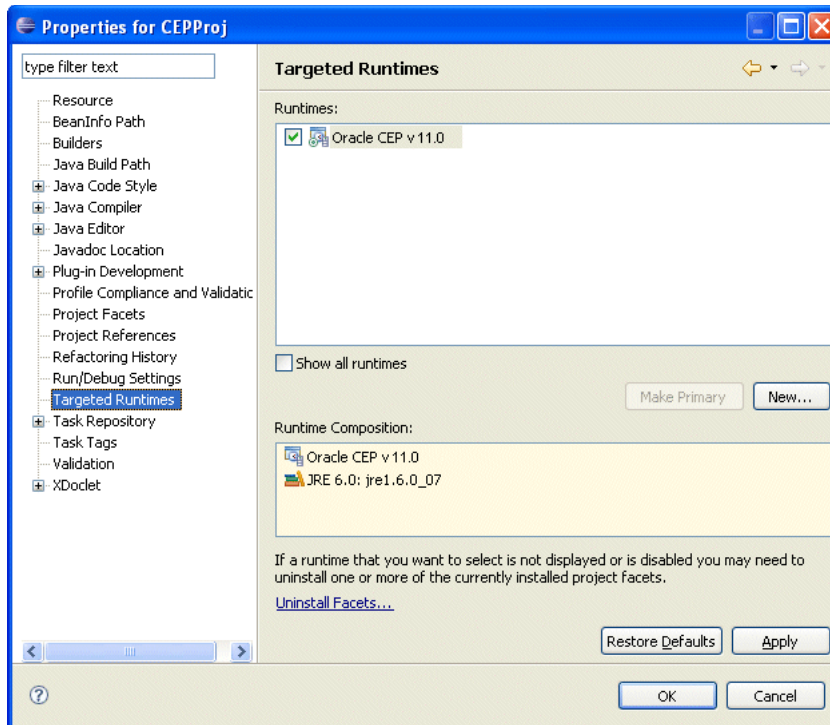
b. Select the **Targeted Runtimes** option.

The Targeted Runtimes properties are displayed as Figure 3–15 shows.

c. Check the new Oracle CEP 11.0 targeted runtime you created.

d. Click **OK**.

e. Repeat for the next project.

11. For each project update the project builders:

a. Right-click the project and select **Open EPN Editor**.

b. If the EPN diagram opens without error, proceed to step 12.

c. If the EPN diagram opens with the error shown in Figure 3–16, click the **Configure Builder** button.

Figure 3–24   Builder Error
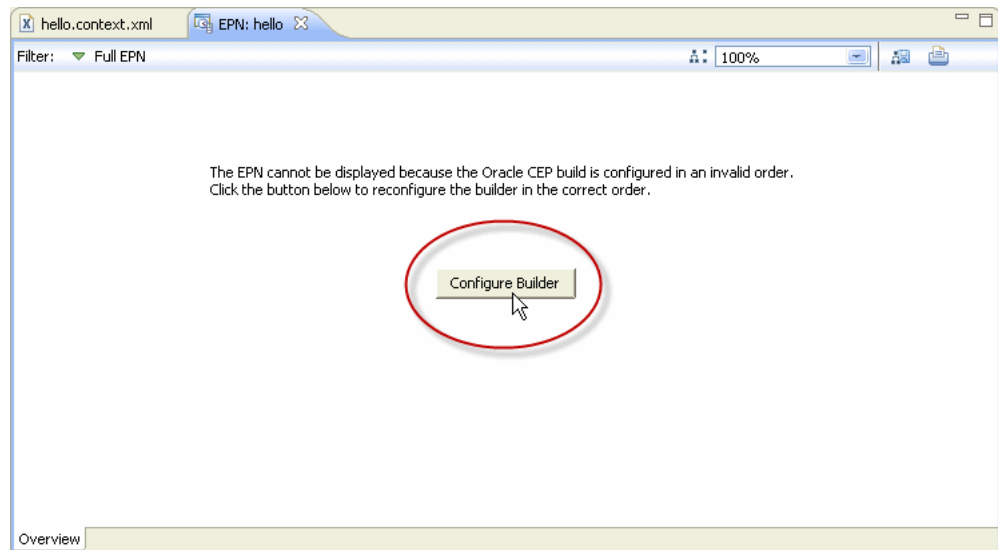
The EPN diagram is displayed.

d. Repeat for the next project.

12. Select **Window > Preferences**.

The Preferences dialog appears as shown in Figure 3–25.

Figure 3–25   Preferences Dialog
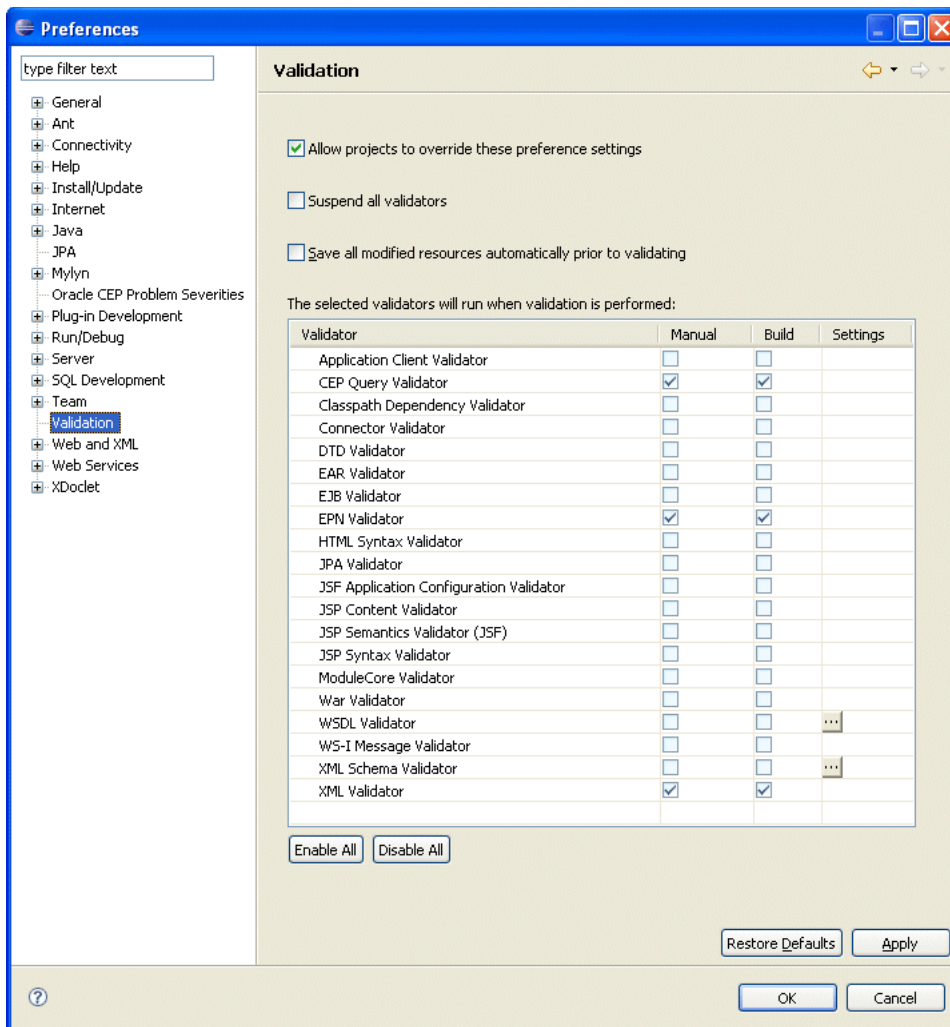
13. Select the **Validaion** option.

14. Ensure that the following validation options are checked:

    ■   CQL Validator

    ■   EPL Validator

    ■   EPN Validator

    ■   XML Validator

15. Unselect all other options.

16. Click **OK**.

17. Validate build inclusions:

    If your application bundle is using bundle localization and has substitution variables in its `MANIFEST.MF` file such as:

    ```
    Bundle-Name: %project.name
    ```

    Then your project root directory's `build.properties` file element `bin.include` must contain a reference to your `bundle.properties` file such as:

```
bin.includes = META-INF/,\
               bundle.properties,\
               .
```

18. Perform source changes, if necessary.

For more information, see:

- "Upgrading an Oracle CEP 10.3 Application to Run on Oracle CEP Release 11gR1 (11.1.1)" in the *Oracle CEP Getting Started*

- *Oracle CEP Release Notes*

3.5 Managing Oracle CEP Project Libraries

Many projects require use of libraries that were obtained from a source other than the project itself, whether that be third party libraries, internal libraries created in other projects, or otherwise.

You can use two types of library in Oracle CEP projects, each with its own packaging and deployment characteristics:

- Standard JAR Files: Adding a standard JAR file to a project makes for the easiest management of the library. The library is packaged directly with the project by the Oracle CEP IDE for Eclipse and you can check the library into a source code control system as part of the project.

  For more information, see Section 3.5.1, "How to Add an Oracle CEP Project Library as a Standard JAR File"

- OSGi Bundles: If your library is already a packaged as an OSGi bundle and you would like to deploy it to the server once (allowing multiple applications to reference it), you can use the OSGi bundle library option. Note that this leaves some parts of deployment to the user since the OSGi bundle is not automatically packaged with the application. It can also make working in team environments a little more difficult because each developer must have the bundle in the `user_projects/modules` directory of their machine, rather than have it source controlled with the rest of the project.

  For more information, see Section 3.5.2, "How to Add an Oracle CEP Project Library as an OSGi Bundle"

3.5.1 How to Add an Oracle CEP Project Library as a Standard JAR File

If the library you need to use is a standard JAR file, you can add it to your Oracle CEP project. Alternatively, you can add a library as an OSGi bundle (see Section 3.5.2, "How to Add an Oracle CEP Project Library as an OSGi Bundle").

To add an Oracle CEP project library as a standard JAR file:

1. Place the JAR file in your Oracle CEP IDE for Eclipse project.

Oracle recommends that you create a folder to put them in such as `lib` as Figure 3–26 shows.

Figure 3–26   Oracle CEP IDE for Eclipse lib Directory
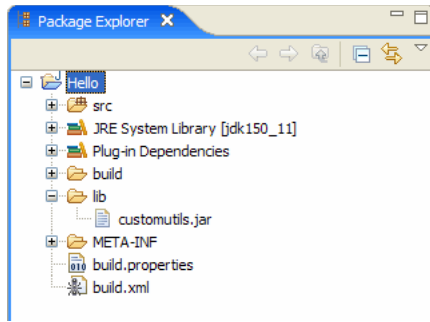
2. Right-click the META-INF/MANIFEST.MF file and select **Open With > Plug-in Manifest Editor**.

The Manifest Editor opens as Figure 3–27 shows.

Figure 3–27   Manifest Editor: Runtime Tab
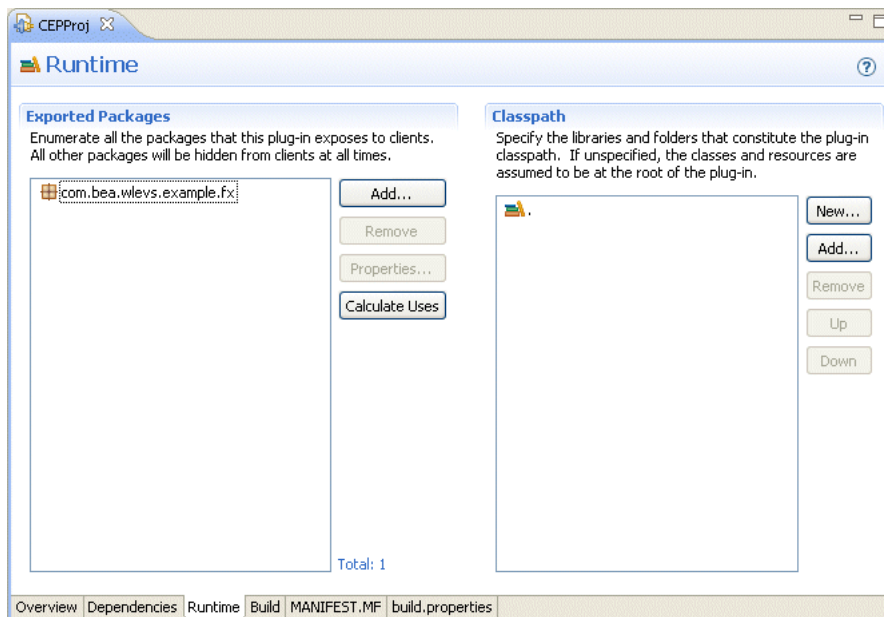
3. Click the **Runtime** tab.

4. In the **Classpath** area, click the **Add** button.

The JAR Selection dialog appears as shown in Figure 3–28.

Figure 3–28   JAR Selection Dialog
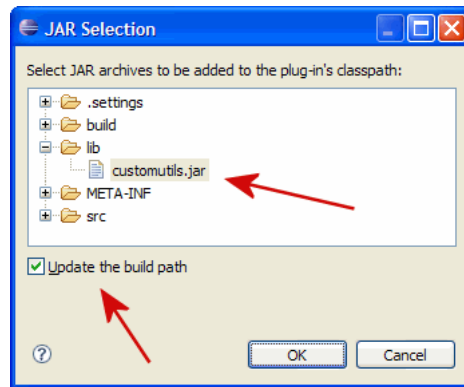
5. Select the JAR you want to add to the bundle.

6. Check **Update the build path**.

7. Click **OK**.

8. Select **File > Save All**.

   You will see the JAR listed in the `Bundle-Classpath` section of the `MANIFEST.MF` file, in the `bin.includes` section of the `build.properties` file, and on the build path in your explorer as Figure 3–29 shows.

Figure 3–29   Package Explorer
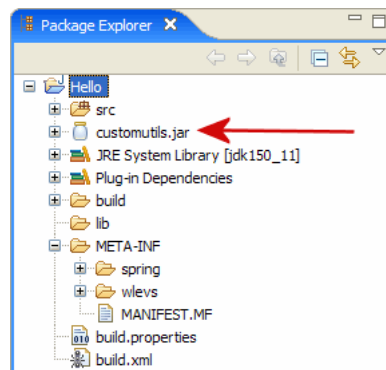
   The JAR file will no longer appear in the `lib` folder in the `Package` and `Project` explorers after you add it to the build path. The JAR will now be included in builds, packaged with the bundle when it is deployed to the Oracle CEP server, and when exporting the application.

3.5.2 How to Add an Oracle CEP Project Library as an OSGi Bundle

If the library you need to use is an OSGi bundle, you can add it to your Oracle CEP project. Alternatively, you can add a library as a standard JAR file (see Section 3.5.1, "How to Add an Oracle CEP Project Library as a Standard JAR File").

To add an Oracle CEP project library as an OSGi bundle, you add the bundle to that bundle's dependencies definition.

To add an Oracle CEP project library as an OSGi bundle:

1. Place the OSGi bundle in the `user_projects/modules` directory of your Oracle CEP server installation directory. For example:

   `c:\bea\user_projects\modules`

2. Start the Oracle CEP IDE for Eclipse.

   refresh the target platform. Navigate to Window > Preferences... > Plug-in Development > Target Platform. On this page, simply hit the "Reload" button. Make sure your bundle now shows in the list then close the preferences dialog.

3. Right-click the `META-INF/MANIFEST.MF` file and select **Open With > Plug-in Manifest Editor**.

   The Manifest Editor opens as Figure 3–30 shows.

Figure 3–30   Manifest Editor: Dependencies Tab
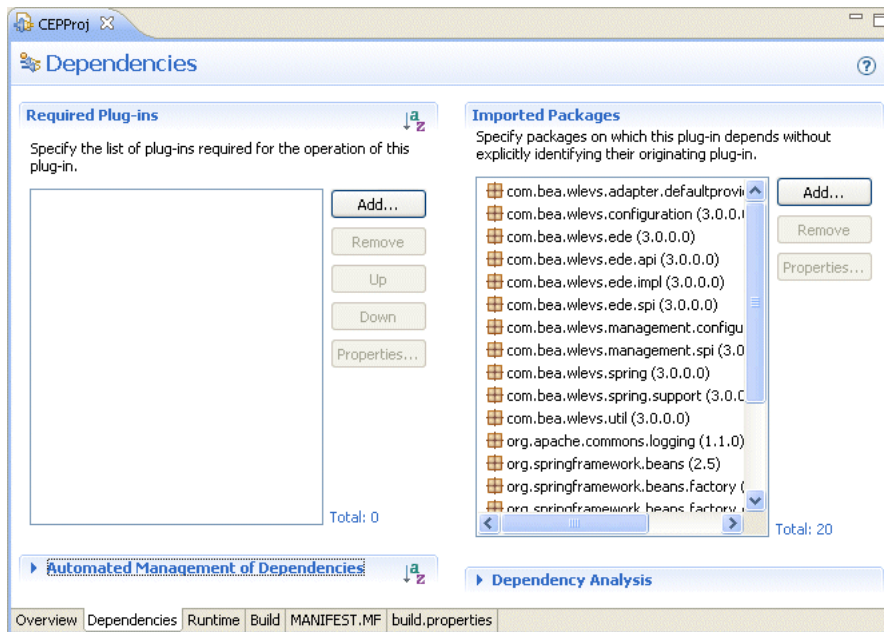
4. Click the **Dependencies** tab.

5. In the **Required Plug-ins** area, click **Add**.

   The Plug-in Selection dialog appears as shown in Figure 3–31.

Configuring Oracle CEP IDE for Eclipse Preferences

Figure 3–31   Plug-in Selection Dialog
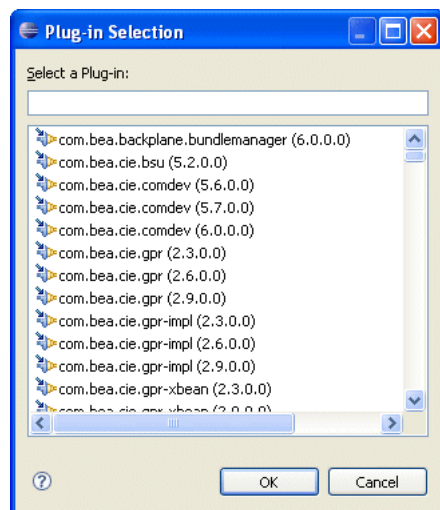
6. Select the bundle you added to the `user_projects/modules` directory of your Oracle CEP server installation directory in step 1 and click **OK**.

    The selected bundle appears in the **Require-Bundle** section of the `MANIFEST.MF` file.

    > **Note:**   This process only makes the referenced bundle available to your project at build time. It does not package the bundle directly with your application when it is deployed or exported. Instead, this bundle must be deployed to the Oracle CEP server manually.

3.6 Configuring Oracle CEP IDE for Eclipse Preferences

You can configure various preferences to customize Oracle CEP IDE for Eclipse to suit your needs, including:

■   Section 3.6.1, "How to Configure Problem Severity Preferences for a Workspace"

■   Section 3.6.2, "How to Configure Problem Severity Properties for a Project"

3.6.1 How to Configure Problem Severity Preferences for a Workspace

You can assign a severity to the various problems that Oracle CEP IDE for Eclipse can detect in your Oracle CEP project and application.

To configure problem severity preferences for a Workspace:

1. Open the EPN Editor (see Section 5.1, "Opening the EPN Editor")

2. Select **Window > Preferences**.

    The Preferences dialog appears as shown in Figure 3–32.

Oracle CEP IDE for Eclipse Projects   **3-29**

Figure 3–32 Preferences Dialog: Workspace
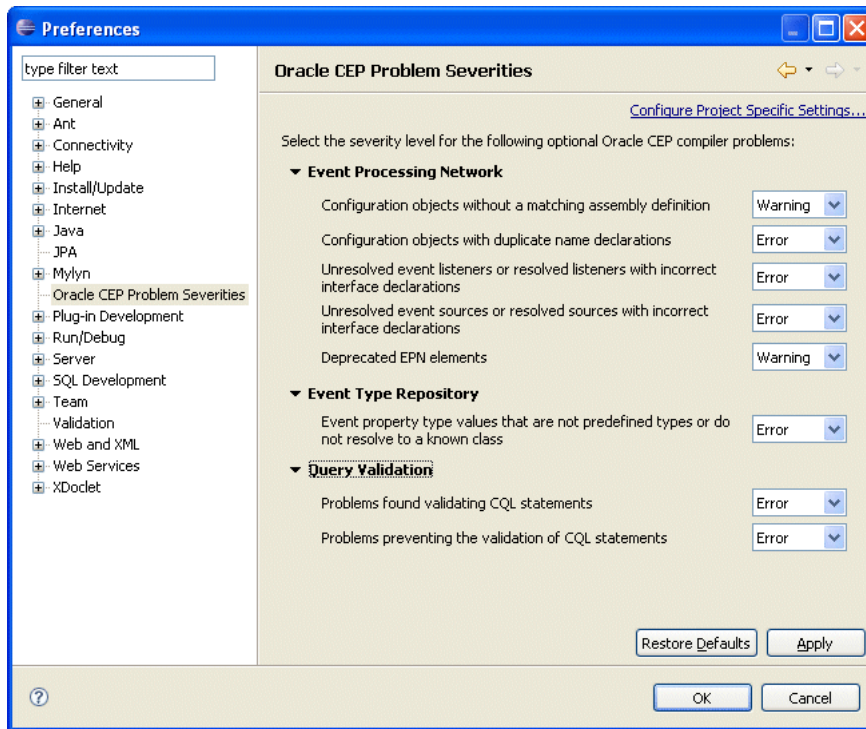
3. Select **Oracle CEP Problem Severities**.

4. Select a severity for each type of problem. You can select one of:

- `Error`: treat the problem as an error.

- `Warning`: treat the problem as a warning.

- `Ignore`: ignore the problem.

Table 3–5 describes each of the problem areas you can configure.

Table 3–5   Oracle CEP Problem Severities

Problem	Description
Configuration objects without a matching assembly definition	EPN configuration elements are linked to assembly definitions by name and ID, respectively. Validate that a configuration element has a name that exactly matches an assembly element by ID within the same application.
Configuration objects with duplicate name declarations.	Configuration elements in Oracle CEP configuration files are identified by a name. Validate that no two configuration elements in an application have the same name.
Unresolved event listeners or resolved listeners with incorrect interface declarations	An event processing network is built by defining how elements in the network pull or push to events to other elements in the application. Validate that the target of an event push, a listener declaration on an EPN assembly element, implements the interfaces required to receive pushed events.
Unresolved event sources or resolved sources with incorrect interface declarations	An event processing network is built by defining how elements in the network pull or push to events to other elements in the application. Validate that the source of an event pull, a source declaration on an EPN assembly element, implements the interfaces required to provide pulled events.
Deprecated EPN elements	Oracle CEP provides backwards compatibility with applications built for previous versions. Validate an application's use of deprecated XML elements.

Table 3–5   (Cont.) Oracle CEP Problem Severities

Problem	Description
Event property type values that are not predefined types or do not resolve to a known class.	Event types may be defined using dynamic Spring Beans through the `properties` element. The `property` values are limited to a fixed set of supported types. Validate that the `property` type is one of these allowed types. For more information, see Section 1.1.2, "Event Types".
Problems found validating CQL Statements	Validate that the Oracle CQL statement in a processor configuration is correct given the current application. Verify property names, event types, syntax, and other assembly-to-Oracle CQL references.
Problems preventing the validation of CQL statements	Some fundamental application errors will keep a Oracle CQL statement from being validated. For example, a processor configuration must have a matching processor assembly definition before any Oracle CQL requirements can be met. Verify that the minimum requirements are met to validate a processor's Oracle CQL statements.

5.   Click **Apply**.

6.   Click **OK**.

3.6.2 How to Configure Problem Severity Properties for a Project

You can assign a severity to the various problems that Oracle CEP IDE for Eclipse can detect in your Oracle CEP project and application.

To configure problem severity properties for a project:

1.   Open the EPN Editor (see Section 5.1, "Opening the EPN Editor").

2.   Right-click a project and select **Properties**.

The Preferences dialog appears as shown in Figure 3–33.

Figure 3–33   Properties Dialog: Project
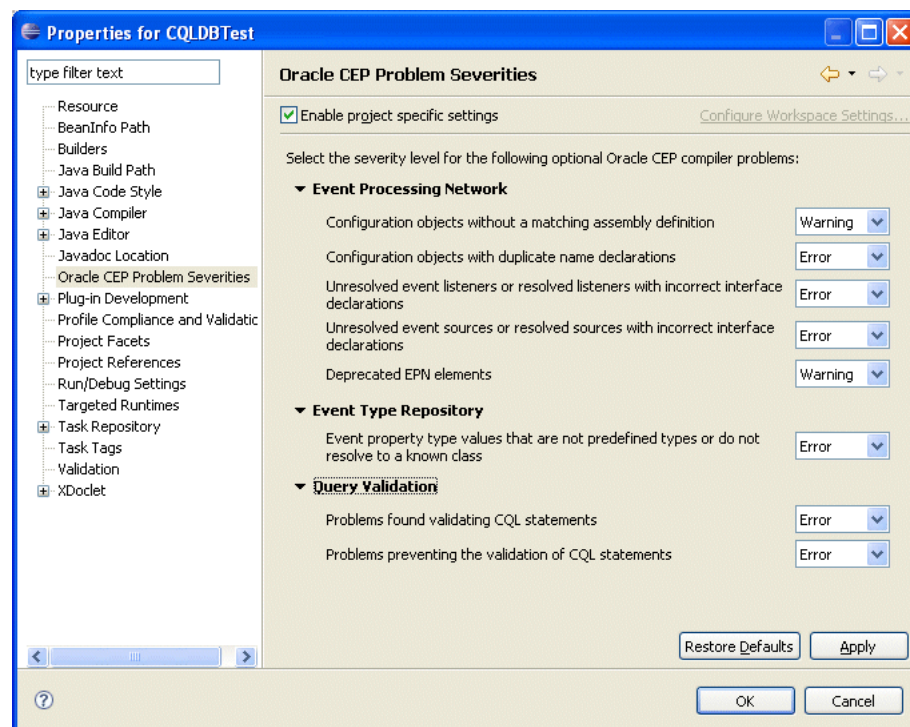
3. Select **Oracle CEP Problem Severities**.

4. Check the **Enable project specific settings** check box.

> **Note:** These properties settings are saved with the project (in the .settings directory) and can be checked into a source code management system.

5. Select a severity for each type of problem. You can select one of:

- `Error`: treat the problem as an error.

- `Warning`: treat the problem as a warning.

- `Ignore`: ignore the problem.

Table 3–6 describes each of the problem areas you can configure.

Table 3–6   Oracle CEP Problem Severities

Problem	Description
Configuration objects without a matching assembly definition	EPN configuration elements are linked to assembly definitions by name and ID, respectively. Validate that a configuration element has a name that exactly matches an assembly element by ID within the same application.
Configuration objects with duplicate name declarations.	Configuration elements in Oracle CEP configuration files are identified by a name. Validate that no two configuration elements in an application have the same name.
Unresolved event listeners or resolved listeners with incorrect interface declarations	An event processing network is built by defining how elements in the network pull or push to events to other elements in the application. Validate that the target of an event push, a listener declaration on an EPN assembly element, implements the interfaces required to receive pushed events.
Unresolved event sources or resolved sources with incorrect interface declarations	An event processing network is built by defining how elements in the network pull or push to events to other elements in the application. Validate that the source of an event pull, a source declaration on an EPN assembly element, implements the interfaces required to provide pulled events.
Deprecated EPN elements	Oracle CEP provides backwards compatibility with applications built for previous versions. Validate an application's use of deprecated XML elements.
Event property type values that are not predefined types or do not resolve to a known class.	Event types may be defined using dynamic Spring Beans through the `properties` element. The `property` values are limited to a fixed set of supported types. Validate that the `property` type is one of these allowed types. For more information, see Section 1.1.2, "Event Types".
Problems found validating CQL Statements	Validate that the Oracle CQL statement in a processor configuration is correct given the current application. Verify property names, event types, syntax, and other assembly-to-Oracle CQL references.
Problems preventing the validation of CQL statements	Some fundamental application errors will keep a Oracle CQL statement from being validated. For example, a processor configuration must have a matching processor assembly definition before any Oracle CQL requirements can be met. Verify that the minimum requirements are met to validate a processor's Oracle CQL statements.

6. Click **Apply**.

7. Click **OK**.

4

Oracle CEP IDE for Eclipse and Oracle CEP Servers

Using Oracle CEP IDE for Eclipse, you can manage and deploy to the Oracle CEP server: the core of an Oracle CEP application. It provides the runtime in which your Oracle CEP application executes.

This chapter describes:

- Section 4.1, "Oracle CEP Server Overview"
- Section 4.2, "Creating Oracle CEP Servers"
- Section 4.3, "Managing Oracle CEP Servers"
- Section 4.4, "Debugging an Oracle CEP Application Running on an Oracle CEP Server"

4.1 Oracle CEP Server Overview

The Oracle CEP IDE for Eclipse provides features that allow you to set up and manage Oracle CEP servers that are used during development. These tools help you to:

- Configure instances of Oracle CEP servers
- Attach to external Oracle CEP server instances
- Manage Oracle CEP server lifecycle with start, stop, and debug commands
- Associate applications with and deploy applications to Oracle CEP servers during development

Table 4–1 maps Eclipse terminology used by the Oracle CEP IDE for Eclipse to Oracle CEP server terminology.

Table 4–1   Eclipse and Oracle CEP Server Concepts

Eclipse IDE Concept	Oracle CEP Server Concept	Description
Runtime	Oracle CEP server installation	The Oracle CEP IDE for Eclipse has the concept of a runtime. The runtime defines the location where the Oracle CEP IDE for Eclipse can find the installation of a particular Oracle CEP server. This information is used to find JAR files and OSGi bundles to add to the project classpath and to further define Servers and Server Instances. Note that a Runtime is not itself a runnable artifact.

Table 4–1 (Cont.) Eclipse and Oracle CEP Server Concepts

Eclipse IDE Concept	Oracle CEP Server Concept	Description
Server and Server Instance	Domain	The Oracle CEP IDE for Eclipse uses the term Server to describe an actual runnable Oracle CEP server instance. You can think of it as something that has start scripts, for example. In Oracle CEP server terminology, this equates to a Domain. When you set up a server, you specify the domain that this instance will run. For more information on domains, see: ■ For more information, see "Creating and Updating Oracle CEP Standalone Server Domains" in the *Oracle CEP Administrator's Guide*. ■ For more information, see "Creating and Updating Oracle CEP Multi-Server Domains" in the *Oracle CEP Administrator's Guide*.
Publish	Deploy	The Oracle CEP IDE for Eclipse typically uses the term Publish to describe physically deploying an application to a server.
Project	Application or Deployment	A project in the Oracle CEP IDE for Eclipse becomes a single Oracle CEP application packaged as an OSGi bundle. It is deployed to a server and shows in the Oracle CEP server's `deployments.xml` file.

Server definitions are the central concept in controlling an Oracle CEP server from the Oracle CEP IDE for Eclipse. It is from the server definition that you start and stop the server. After associating a project with the server, you can publish (deploy) the application to and unpublish (undeploy) the application from the server, all without having to leave the Oracle CEP IDE for Eclipse. For more information, see Section 4.2, "Creating Oracle CEP Servers".

You can communicate with a running Oracle CEP server using Oracle CEP IDE for Eclipse in the following ways:

■ Start a server from within Oracle CEP IDE for Eclipse.

In this case, the Oracle CEP server console is sent directly to the console view in Oracle CEP IDE for Eclipse. All of the Oracle CEP server features (such as start, stop, publish, unpublish, debug, and launching the Oracle CEP Visualizer) are available. The Oracle CEP server process itself is managed from within Oracle CEP IDE for Eclipse. In other words, stopping the Oracle CEP server from the Oracle CEP IDE for Eclipse will terminate the actual Oracle CEP server process. Console messages from the Oracle CEP server are sent to the Oracle CEP IDE for Eclipse Console view.

For more information, see:

– Section 4.3.1, "How to Start an Oracle CEP Server"

– Section 4.3.2, "How to Stop an Oracle CEP Server"

■ Attach to a running Oracle CEP server.

In this case, the user starts the Oracle CEP server from the command line, then clicks the **Start** button for that server in Oracle CEP IDE for Eclipse. A dialog is shown asking whether or not to attach and, if the user clicks **Yes**, Oracle CEP IDE for Eclipse enters attached mode. All of the Oracle CEP server features except debug are available. However, the Oracle CEP server process is not managed by the Oracle CEP IDE for Eclipse. Clicking the **Stop** button simply disconnects from the attached Oracle CEP server; it does not terminate the actual Oracle CEP server process. Console messages from the Oracle CEP server are sent to the Oracle CEP server console (standard output to the terminal window in which it is running).

Oracle CEP IDE for Eclipse only shows limited Oracle CEP IDE for Eclipse operation messages in the console view.

For more information, see:

– Section 4.3.3, "How to Attach to an Existing Oracle CEP Server Instance"

– Section 4.3.4, "How to Detach From an Existing Oracle CEP Server Instance"

4.2 Creating Oracle CEP Servers

Creating a server allows you to start and stop the server instance from within the Oracle CEP IDE for Eclipse, as well as automatically deploy your applications to that server.

This section describes:

■ Section 4.2.1, "How to Create an Oracle CEP Server Runtime"

■ Section 4.2.2, "How to Create an Oracle CEP Server and Server Runtime"

4.2.1 How to Create an Oracle CEP Server Runtime

Before you can create a server, you must configure the Oracle CEP IDE for Eclipse with the location of your Oracle CEP server installation by creating a server runtime using the runtime wizard. You can access the runtime wizard from several places including the new server wizard, the new project wizard, and the workspace preferences dialog.

You only need to create a runtime explicitly if you have not yet created an Oracle CEP server (see Section 4.2.2, "How to Create an Oracle CEP Server and Server Runtime").

To create an Oracle CEP server runtime:

1. Select **Windows > Preferences**.

   The Preferences dialog appears as Figure 4–1 shows.

Figure 4–1   Preferences - Server - Installed Runtimes
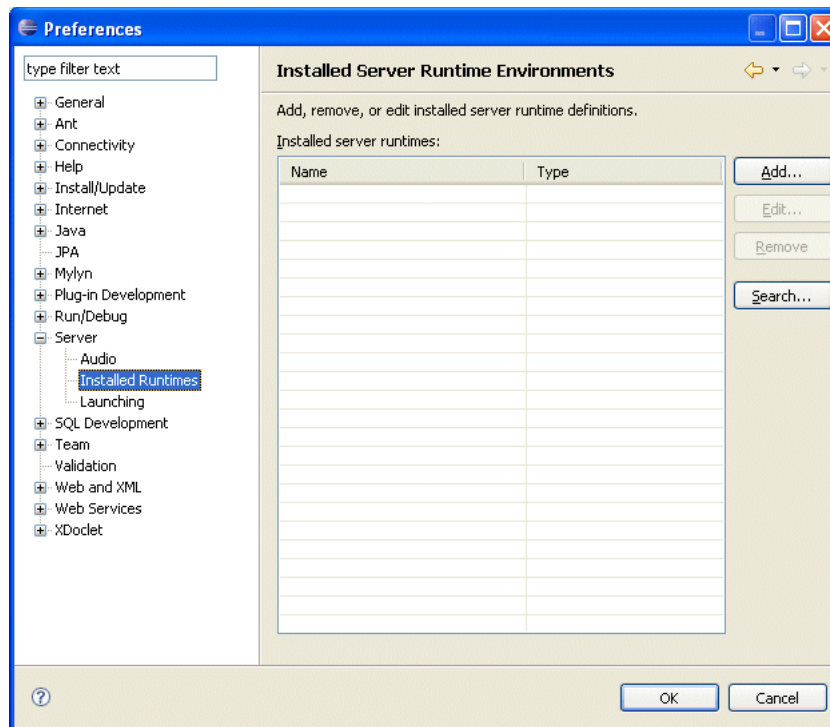
2. Expand the **Server** option and select **Installed Runtimes**.

3. Click **Add**.

   The New Server Runtime dialog appears as shown in Figure 4–2.

Figure 4–2   New Server Runtime Dialog
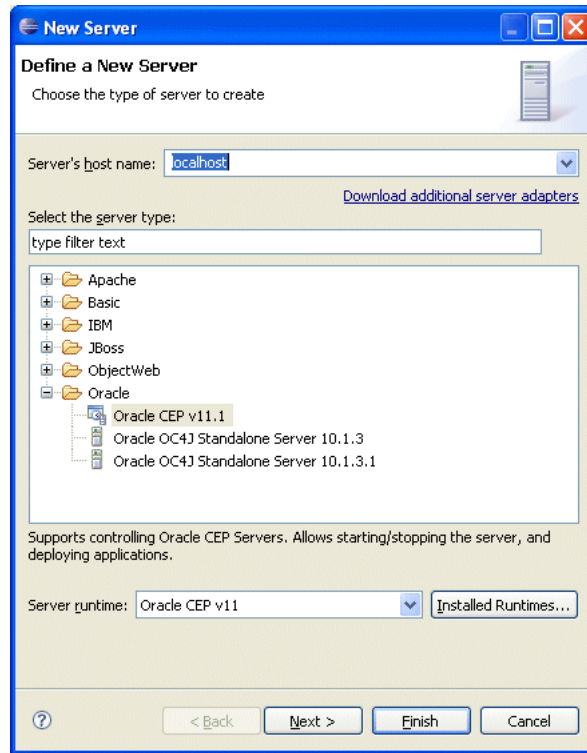
4. Configure the dialog as shown in Table 4–2.

Table 4–2   New Server Runtime Dialog Attributes

Attribute	Description
Server's host name:	The host name of the computer on which you installed Oracle CEP server.
	For development, this will typically be `localhost`.
Select the server type:	The type of Oracle CEP server.
	In this example, choose `Oracle CEP v11.1`.
Also create new local server	Optionally, check this to create a new local server if you have not yet created a server. For more information, see Section 4.2.2, "How to Create an Oracle CEP Server and Server Runtime".

5. Click **Next**.

   The New Server: New Oracle CEP v11 Runtime dialog appears as shown in Figure 4–3.

Figure 4–3   New Server: New Oracle CEP v11 Runtime Dialog
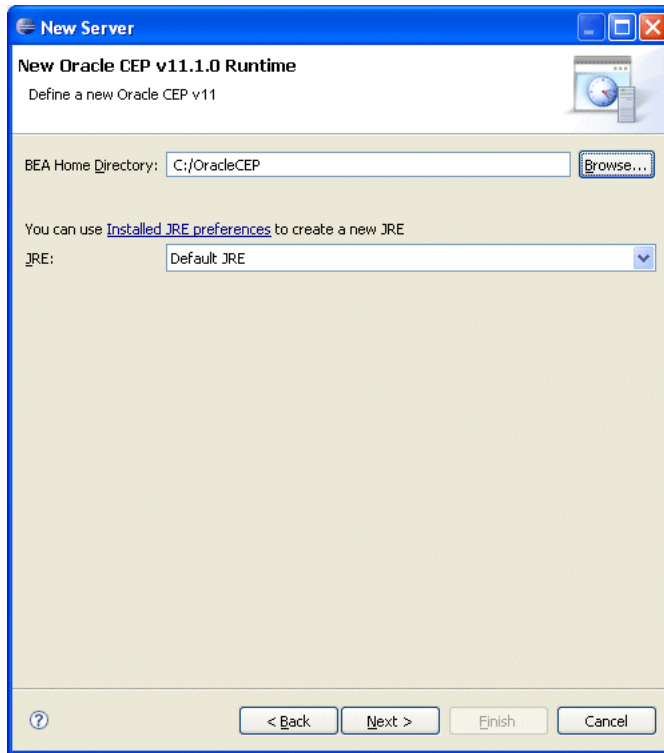
6. Configure the dialog as shown in Table 4–3.

Table 4–3   New Server Runtime Dialog Attributes

Attribute	Description

BEA Home Directory	The fully qualified path to the Oracle CEP server installation directory. This is the same as the "Middleware Home" that was selected when installing the server.
	
	When selecting this directory, select directory that contains the Oracle CEP installation rather than the Oracle CEP directory itself. For example, choose:
	
	`C:\OracleCEP`
	
	But do not choose:
	
	`C:\OracleCEP\ocep_11.1`
	
	The runtime wizard will use the installation to find the appropriate Oracle CEP installation directory.

JRE	The type of JRE to use. Select the type of JRE to use from the pull-down menu or click the **Installed JRE preferences** link to create a new JRE. Be sure to choose either JRockit Real Time 2.0 or the JRockit JDK installed with your Oracle CEP installation.

7. Click **Finish**.

4.2.2 How to Create an Oracle CEP Server and Server Runtime

This section describes how to create both a server and server runtime. After creating the initial server and server runtime, you can create additional server runtimes (see Section 4.2.1, "How to Create an Oracle CEP Server Runtime").

To create an Oracle CEP server and Server Runtime:

1. Select **Window > Show View > Servers**.

   The Servers view appears as shown in Figure 4–4.

Figure 4–4   Oracle CEP IDE for Eclipse Server View
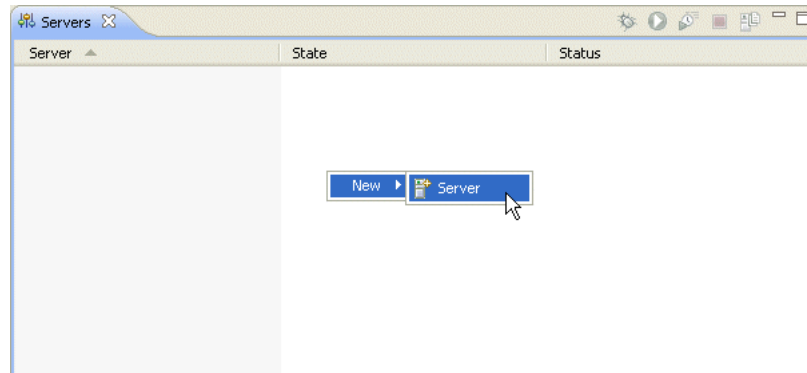
2. Right-click in the **Servers** view pane and select **New > Server**.

3. If this is the first time you have created an Oracle CEP server, there will be no installed server runtimes:

   In this case, the New Server: Define New Server dialog appears as Figure 4–5 shows.

Figure 4–5   New Server: Define New Server Dialog (No Installed Runtimes)
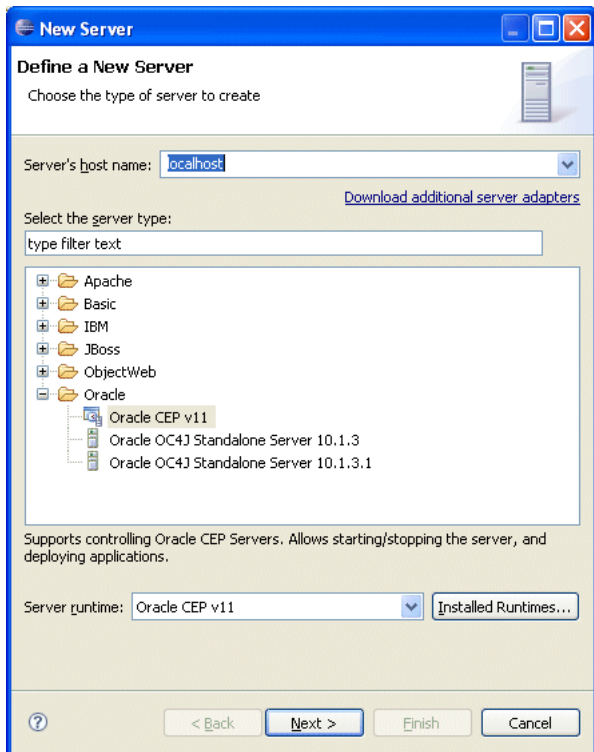
Configure the new server as follows:

a.   Configure the dialog as shown in Table 4–4.

Table 4–4   New Server: Define New Server Dialog (No Installed Runtimes) Attributes

Attribute	Description
Server's host name:	The host name of the computer on which you installed Oracle CEP server. For development, this will typically be `localhost`.
Select the server type:	The type of Oracle CEP server. In this example, choose `Oracle CEP v11`

b.   Click **Next**.

The New Server: New Oracle CEP v11 Runtime dialog appears as shown in Figure Figure 4–6.

Figure 4–6   New Server: New Oracle CEP v11 Runtime Dialog
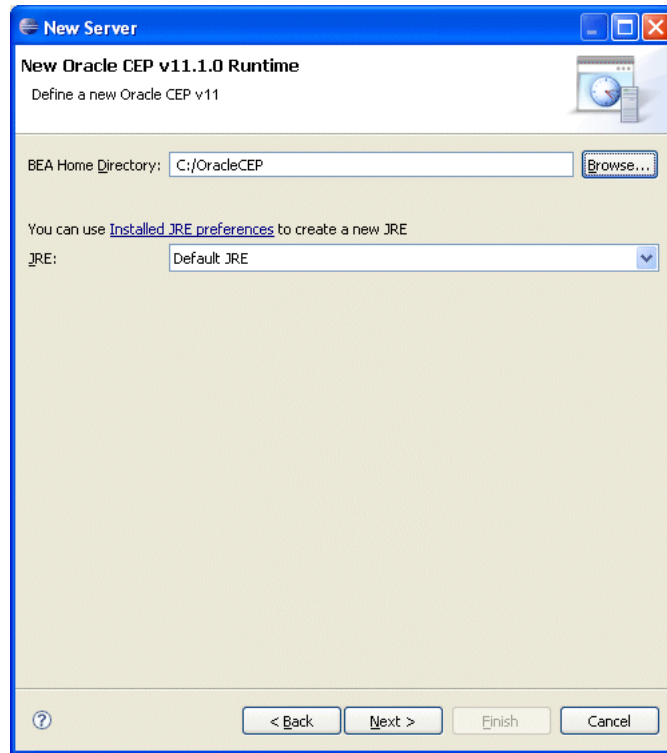
c.   Configure the dialog as shown in Table 4–5.

Table 4–5   New Server: New Oracle CEP v11 Runtime Dialog Attributes

Attribute	Description

BEA Home Directory	The fully qualified path to the Oracle CEP server installation directory.
	
	When selecting this directory, select directory that contains the Oracle CEP installation rather than the Oracle CEP directory itself. For example, choose:
	
	`C:\OracleCEP`
	
	But do not choose:
	
	`C:\OracleCEP\ocep_11.1`
	
	The runtime wizard will use the installation to find the appropriate Oracle CEP installation directory.

JRE	The type of JRE to use. Select the type of JRE to use from the pull-down menu or click the **Installed JRE preferences** link to create a new JRE. Be sure to choose either JRockit Real Time 2.0 or the JRockit JDK installed with your Oracle CEP installation.

d.   Click **Next**.

e.   Proceed to step 5.

4.   If this is not the first time you have created an Oracle CEP server, there will be one or more installed server runtimes.

In this case, the New Server: Define New Server dialog appears as Figure 4–7 shows.

Figure 4–7   New Server: Define New Server (Installed Runtimes) Dialog
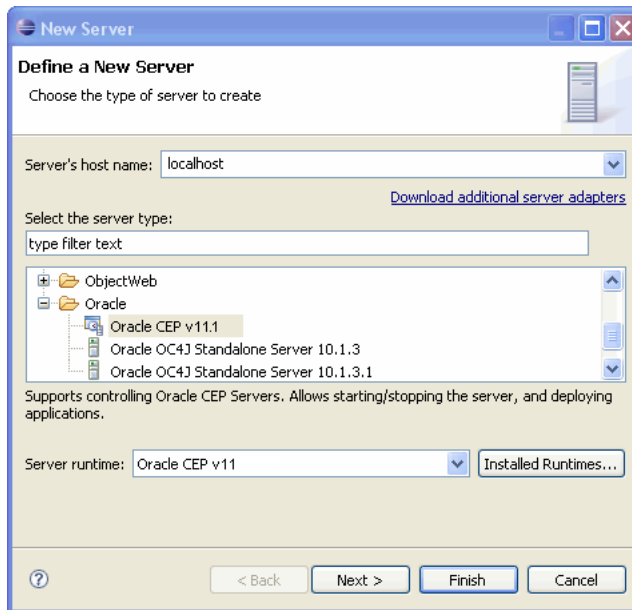
Configure the new server as follows:

a. Configure the dialog as shown in Table 4–6.

Table 4–6   New Server: Define New Server (Installed Runtimes) Dialog Attributes

Attribute	Description
Server's host name:	The host name of the computer on which you installed Oracle CEP server. For development, this will typically be `localhost`.
Select the server type:	The type of Oracle CEP server. In this example, choose `Oracle CEP v11`.
Server runtime:	Select the server runtime from the pull-down menu. To create or edit server runtimes, click **Installed Runtimes**. For more information, see Section 4.2.1, "How to Create an Oracle CEP Server Runtime".

b. Click **Next**.

5. The New Server: New Oracle CEP v11 Server dialog appears as Figure 4–8 shows.

Figure 4–8   New Server: New Oracle CEP v11 Server Dialog
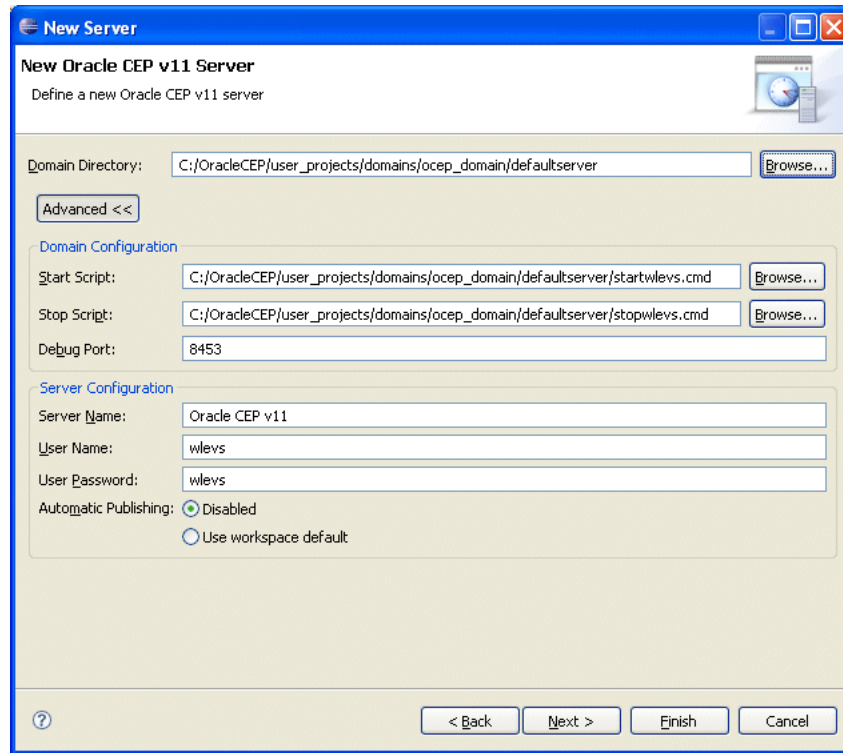
6. Click **Advanced**.

7. Configure the dialog as shown in Table 4–7.

Table 4–7   New Server: New Oracle CEP v11 Server Dialog Attributes

Attribute	Description
Domain Directory	The fully qualified path to the directory that contains the domain for this server.
	Click **Browse** to choose the directory.
	Default: `ORACLE-CEP-HOME\user_projects\domains\ocep_ domain\defaultserver`.
Start Script	The script that Oracle CEP IDE for Eclipse uses to start the Oracle CEP server.
	Default on UNIX: `ORACLE-CEP-HOME/user_ projects/domains/ocep_ domain/defaultserver/stopwlevs.sh`
	Default on Windows: `ORACLE-CEP-HOME\user_ projects\domains\ocep_ domain\defaultserver\stopwlevs.cmd`
Stop Script	The script that Oracle CEP IDE for Eclipse uses to stop the Oracle CEP server.
	Default on UNIX: `ORACLE-CEP-HOME/\user_ projects/domains/ocep_ domain/defaultserver/startwlevs.sh`
	Default on Windows: `ORACLE-CEP-HOME\user_ projects\domains\ocep_ domain\defaultserver\startwlevs.cmd`

Table 4–7   (Cont.) New Server: New Oracle CEP v11 Server Dialog Attributes

Attribute	Description
Debug Port	The Oracle CEP server port that Oracle CEP IDE for Eclipse connects to when debugging the Oracle CEP server. Default: 8453.
Server Name	The name of the Oracle CEP server. Default: Oracle CEP v11
User Name	The user name Oracle CEP IDE for Eclipse uses when logging into the Oracle CEP server. Default: `wlevs`.
User Password	The user password Oracle CEP IDE for Eclipse uses when logging into the Oracle CEP server. Default: `wlevs`.
Automatic Publishing	By default, when you change an application, you must manually publish the changes to the Oracle CEP server. Select **Use Workspace Default** to configure Oracle CEP IDE for Eclipse to automatically publish changes to the Oracle CEP server. Default: `Disabled`.

8. Click **Finish**.

9. If you configured **Automatic Publishing** as **Use Workspace Default**, select **Windows > Preferences**.

    The Preferences dialog appears as Figure 4–9 shows.

Figure 4–9   Preferences - Server
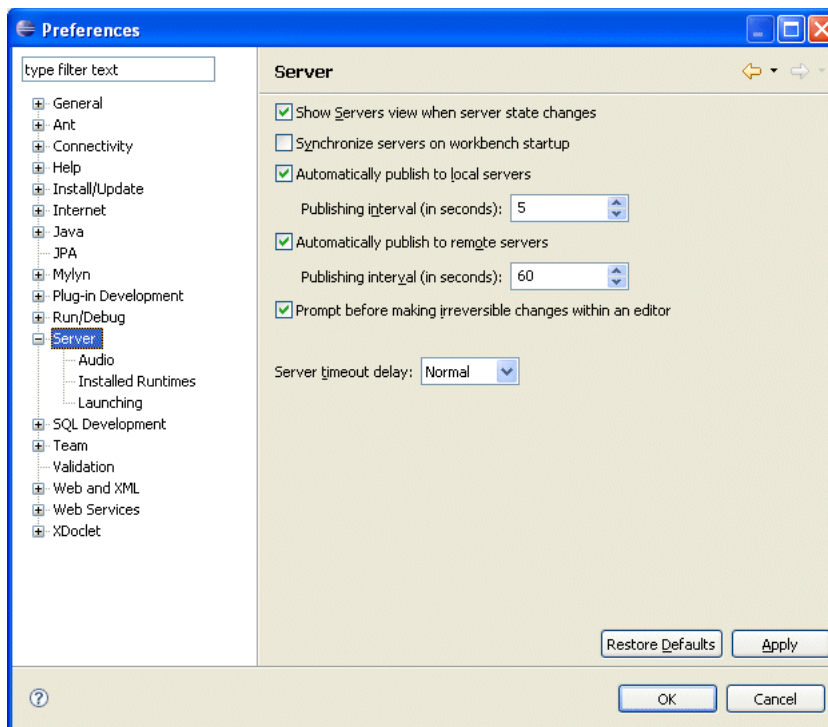
10. Select the **Server** option.

11. Configure your automatic publishing options:

   ■ **Automatically publish to local servers**: enable or disable this option, as required.

      Default: enabled.

      – **Publishing interval**: configure the frequency at which the Oracle CEP IDE for Eclipse publishes changes to the server (in seconds).

         Default: 60 seconds.

   ■ **Automatically publish to remote servers**: enable or disable this option, as required.

      Default: enabled.

      – **Publishing interval**: configure the frequency at which the Oracle CEP IDE for Eclipse publishes changes to the server (in seconds).

         Default: 60 seconds.

12. Click **OK**.

4.3 Managing Oracle CEP Servers

Using the Oracle CEP IDE for Eclipse and the Oracle CEP Visualizer accessible from the Oracle CEP IDE for Eclipse, you can manage many aspects of your Oracle CEP server during development.

This section describes the following Oracle CEP server management tasks you can perform from the Oracle CEP IDE for Eclipse:

■ Section 4.3.1, "How to Start an Oracle CEP Server"

■ Section 4.3.2, "How to Stop an Oracle CEP Server"

■ Section 4.3.3, "How to Attach to an Existing Oracle CEP Server Instance"

■ Section 4.3.4, "How to Detach From an Existing Oracle CEP Server Instance"

■ Section 4.3.5, "How to Deploy an Application to an Oracle CEP Server"

■ Section 4.3.6, "How to Configure Connection and Control Settings for Oracle CEP Server"

■ Section 4.3.7, "How to Configure Domain (Runtime) Settings for Oracle CEP Server"

■ Section 4.3.8, "How to Start the Oracle CEP Visualizer from Oracle CEP IDE for Eclipse"

4.3.1 How to Start an Oracle CEP Server

After you create a server (see Section 4.2.2, "How to Create an Oracle CEP Server and Server Runtime"), you can start the Oracle CEP server from the Oracle CEP IDE for Eclipse.

You can also start the Oracle CEP server in debug mode (see Section 4.4.1, "How to Debug an Oracle CEP Application Running on an Oracle CEP Server").

Alternatively, you can start the Oracle CEP server from the command line and attach to it using Oracle CEP IDE for Eclipse (see Section 4.3.3, "How to Attach to an Existing Oracle CEP Server Instance").

To start an Oracle CEP server:

1. Select **Window > Show Views > Servers**.

   The Servers view opens as shown in Figure 4–10.

Figure 4–10   Starting an Oracle CEP Server
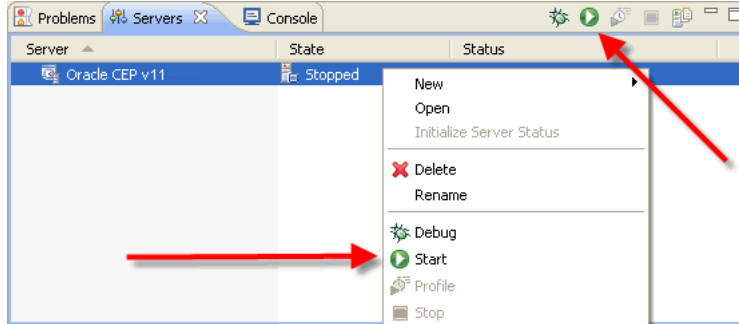
2. Start the server by choosing one of the following:

   a. Click the **Start the Server** icon in the Servers view tool bar.

   b. Right-click a server in the Servers view and select **Start**.

   After starting the server you will see log messages from the server in the Console view.

4.3.2 How to Stop an Oracle CEP Server

After you start the Oracle CEP server from the Oracle CEP IDE for Eclipse (see Section 4.3.1, "How to Start an Oracle CEP Server"), you can stop the Oracle CEP server from the Oracle CEP IDE for Eclipse.

To stop an Oracle CEP server:

1. Select **Window > Show Views > Servers**.

   The Servers view opens as shown in Figure 4–11.

Figure 4–11   Stopping an Oracle CEP Server
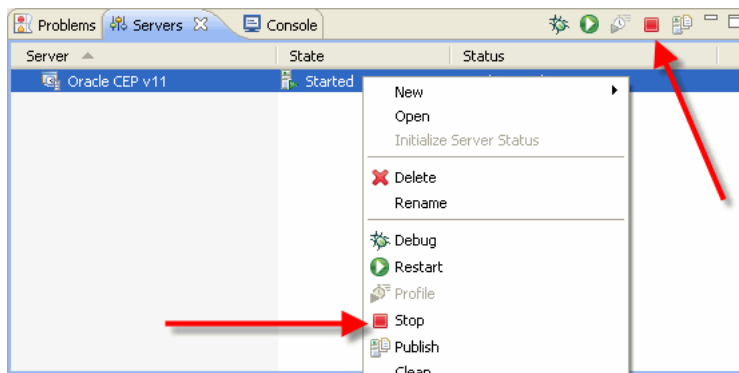
2. Stop the server by choosing one of the following:

   a. Click the **Stop the Server** icon in the Servers view tool bar.

   b. Right-click a server in the Servers view and select **Stop**.

4.3.3 How to Attach to an Existing Oracle CEP Server Instance

After you create a server (see Section 4.2.2, "How to Create an Oracle CEP Server and Server Runtime"), you can start the Oracle CEP server from the command line and attach Oracle CEP IDE for Eclipse to the existing, already running Oracle CEP server instance.

Alternatively, you can start the Oracle CEP server directly from within Oracle CEP IDE for Eclipse (see Section 4.3.1, "How to Start an Oracle CEP Server").

To attach to an existing Oracle CEP server instance:

1. Start the Oracle CEP server from the command line.

For more information, see "Starting and Stopping Oracle CEP Servers" in the *Oracle CEP Administrator's Guide*.

2. Select **Window > Show Views > Servers**.

The Servers view opens as shown in Figure 4–10.

Figure 4–12   Attaching to an Existing Oracle CEP Server Instance
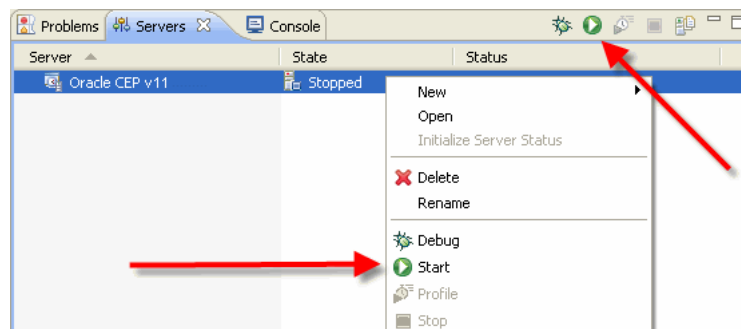
3. Attach to the already running server by choosing one of the following:

a. Click the **Start the Server** icon in the Servers view tool bar.

b. Right-click a server in the Servers view and select **Start**.

The Attach to Running CEP Server dialog appears as Figure 4–13 shows.

Figure 4–13   Attach to Running CEP Server
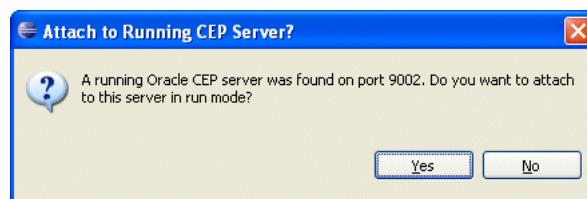
4. Click **Yes**.

After attaching to the server you will *not* see log messages from the server in the Console view.

You can view the server console using the Oracle CEP Visualizer. For more information, see "How to View Console Output" in the *Oracle CEP Visualizer User's Guide*.

4.3.4 How to Detach From an Existing Oracle CEP Server Instance

After you attach to an existing, running Oracle CEP server instance (see Section 4.3.3, "How to Attach to an Existing Oracle CEP Server Instance"), you can detach from the Oracle CEP server and leave it running.

To detach from an existing Oracle CEP server instance:

1. Select **Window > Show Views > Servers**.

    The Servers view opens as shown in Figure 4–11.

Figure 4–14 Stopping an Oracle CEP Server
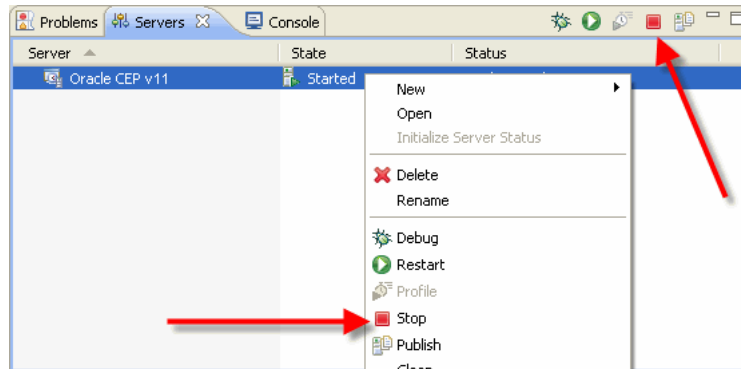
2. Detach from the server by choosing one of the following:

    a. Click the **Stop the Server** icon in the Servers view tool bar.

    b. Right-click a server in the Servers view and select **Stop**.

    Oracle CEP IDE for Eclipse detaches from the Oracle CEP server instance. The Oracle CEP server instance continues to run.

4.3.5 How to Deploy an Application to an Oracle CEP Server

A project in the Oracle CEP IDE for Eclipse is built as an Oracle CEP application, then deployed to the server. To deploy an application, a server must first be defined (see Section 4.2.2, "How to Create an Oracle CEP Server and Server Runtime"). To then deploy an application, simply add it to the server. The application will be deployed immediately if the server is already started, or when the server is next started if the server is stopped.

To deploy an application to an Oracle CEP server:

1. Create an Oracle CEP project (see Section 3.2, "Creating Oracle CEP Projects").

2. Create a server (see Section 4.2.2, "How to Create an Oracle CEP Server and Server Runtime").

3. Select **Window > Show Views > Servers**.

    The Servers view opens as shown in Figure 4–15.

Figure 4–15   Adding a Project to an Oracle CEP Server
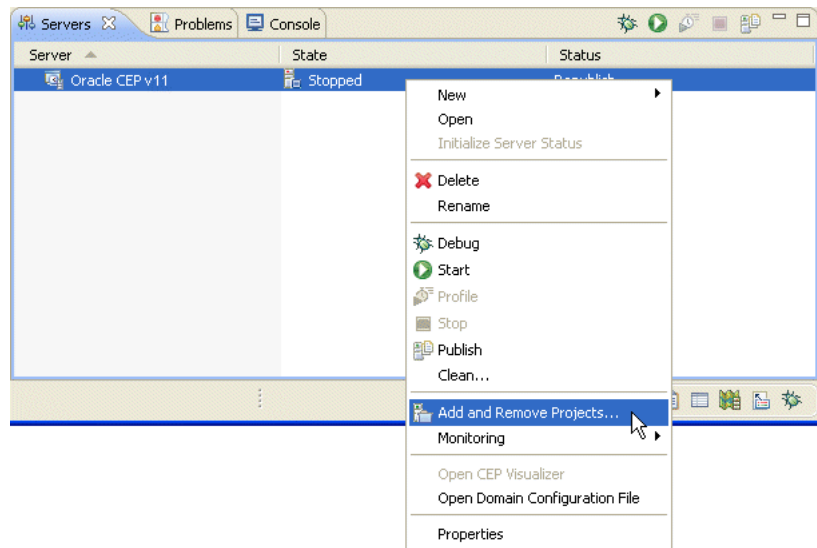
4. Right-click the server and select **Add and Remove Projects**.

The Add and Remove Projects dialog appears as shown in Figure 4–16.

Figure 4–16   Add and Remove Projects Dialog
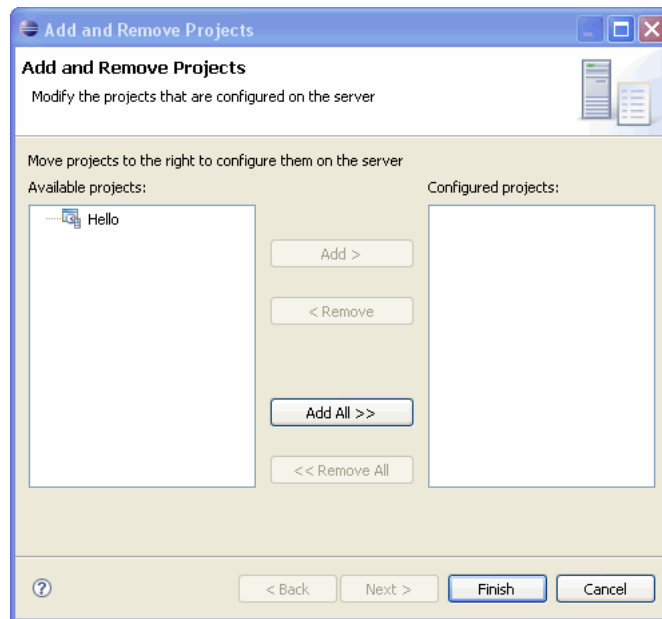
5. Select a project and click **Add** or click **Add All** to add all projects to the Oracle CEP server.

6. Click **Finish**.

Once an application is deployed, it will show as a child of the server in the Servers view as shown in Figure 4–17.

Figure 4–17   Server View After Adding a Project
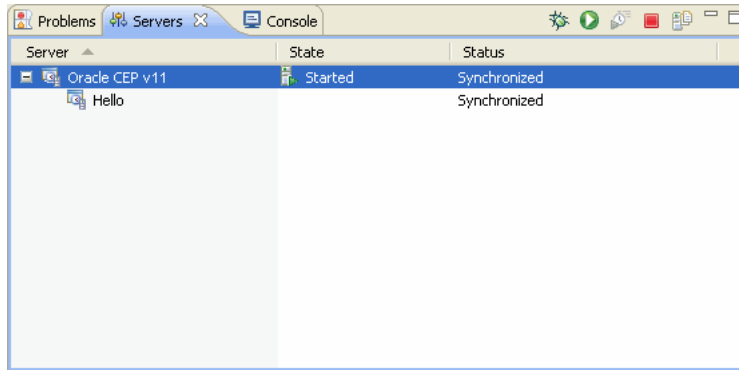
4.3.6 How to Configure Connection and Control Settings for Oracle CEP Server

After you create a server (see Section 4.2.2, "How to Create an Oracle CEP Server and Server Runtime"), you can use the Server Overview editor to configure all the important server connection and control settings that Oracle CEP IDE for Eclipse uses to communicate with the Oracle CEP server.

For information on how to configure Oracle CEP server domain (runtime) settings, see Section 4.3.7, "How to Configure Domain (Runtime) Settings for Oracle CEP Server".

To configure connection and control settings for Oracle CEP server:

1. Select **Window > Show Views > Servers**.

2. Double-click a server in the Servers view.

   The Server Overview editor opens as shown in Figure 4–18.

Figure 4–18   Server Overview Editor
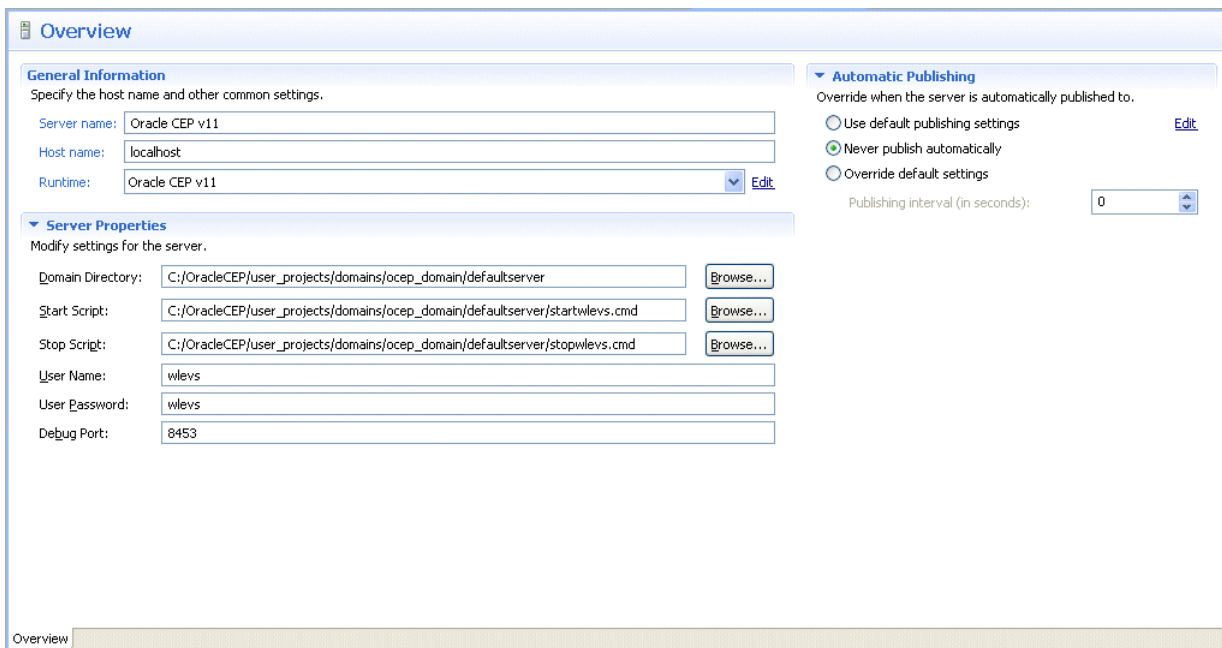
3. Configure the Server Overview editor as shown in Table 4–8.

Table 4–8   Server Overview Editor Attributes

Attribute	Description
Server Name	The name of this server. Only used within the Oracle CEP IDE for Eclipse as a useful identifier.
	For more information, see Section 4.2.2, "How to Create an Oracle CEP Server and Server Runtime".
Host Name	The name of the host on which this server is installed.
	For more information, see Section 4.2.2, "How to Create an Oracle CEP Server and Server Runtime".
Runtime	The current installed runtime selected for this server.
	Select a new runtime from the pull down menu or click the **Edit** link to modify the configuration of the selected runtime.
	For more information, see Section 4.2.2, "How to Create an Oracle CEP Server and Server Runtime".
Domain Directory	The fully qualified path to the directory that contains the domain for this server.
	Click **Browse** to choose the directory.
	Default: *ORACLE-CEP-HOME*\user_ projects\domains\ocep_domain\defaultserver.
Start Script	The script that Oracle CEP IDE for Eclipse uses to start the Oracle CEP server.
	Click **Browse** to choose the start script.
	Default on UNIX: *ORACLE-CEP-HOME*\user_ projects\domains\ocep_ domain\defaultserver\stopwlevs.sh
	Default on Windows: *ORACLE-CEP-HOME*\user_ projects\domains\ocep_ domain\defaultserver\stopwlevs.cmd
Stop Script	The script that Oracle CEP IDE for Eclipse uses to stop the Oracle CEP server.
	Click **Browse** to choose the stop script.
	Default on UNIX: *ORACLE-CEP-HOME*\user_ projects\domains\ocep_ domain\defaultserver\startwlevs.sh
	Default on Windows: *ORACLE-CEP-HOME*\user_ projects\domains\ocep_ domain\defaultserver\startwlevs.cmd
Debug Port	The Oracle CEP server port that Oracle CEP IDE for Eclipse connects to when debugging the Oracle CEP server.
	Default: 8453.
Server Name	The name of the Oracle CEP server.
	Default: `Oracle CEP v11`
User Name	The user name Oracle CEP IDE for Eclipse uses when logging into the Oracle CEP server.
	Default: `wlevs`.
User Password	The user password Oracle CEP IDE for Eclipse uses when logging into the Oracle CEP server.
	Default: `wlevs`.

Table 4–8   (Cont.) Server Overview Editor Attributes

Attribute	Description
Override when the server is automatically published to	By default, when you change an application, you must manually publish the changes to the Oracle CEP server.
	Select **Use default publishing settings** to enable automatic publishing using the default publishing preferences. Click the **Edit** link to configure the default publishing preferences.
	Select **Never publish automatically** to disable automatic publishing.
	Select **Override default settings** to override the default automatic publishing interval. Enter a new publishing interval (in seconds).
	Default: `Never publish automatically.`

4. Select **File > Save**.

5. Close the Server Overview editor.

4.3.7 How to Configure Domain (Runtime) Settings for Oracle CEP Server

After you create a server (see Section 4.2.2, "How to Create an Oracle CEP Server and Server Runtime"), you can use the Oracle CEP IDE for Eclipse to configure Oracle CEP server domain (runtime) settings.

You can also use the Oracle CEP IDE for Eclipse to configure all the important server connection and control settings that Oracle CEP IDE for Eclipse uses to communicate with the Oracle CEP server (see Section 4.3.6, "How to Configure Connection and Control Settings for Oracle CEP Server").

To configure domain (runtime) settings for Oracle CEP server:

1. Select **Window > Show Views > Servers**.

2. Right-click a server in the Servers view and select **Open Domain Configuration File** as shown in Figure 4–19.

Figure 4–19   Editing the Domain Configuration File
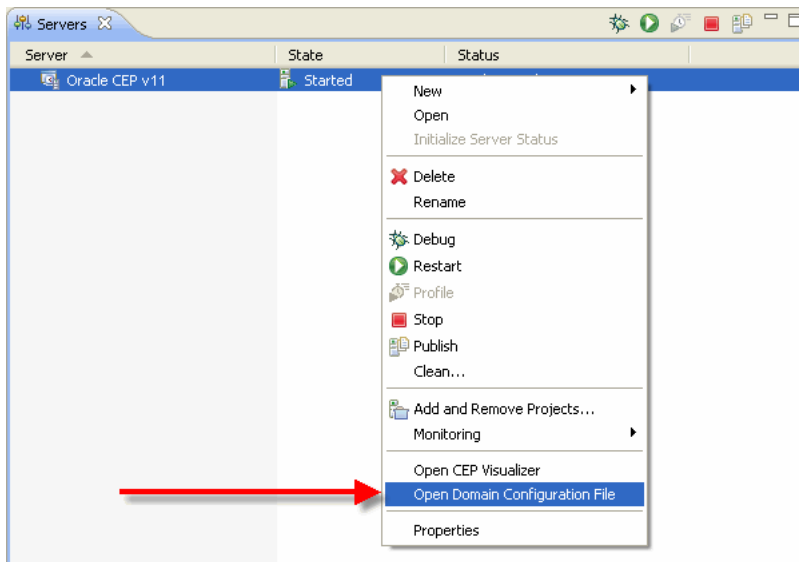
The Oracle CEP server domain configuration file `config.xml` opens as shown in Figure 4–20.

Figure 4–20 Oracle CEP Domain Configuration File config.xml

3. Edit the domain configuration file as required.

4. Select **File > Save**.

5. Close the domain configuration file.

4.3.8 How to Start the Oracle CEP Visualizer from Oracle CEP IDE for Eclipse

After you create a server (see Section 4.2.2, "How to Create an Oracle CEP Server and Server Runtime"), you can start the Oracle CEP Visualizer from the Oracle CEP IDE for Eclipse.

The Oracle CEP Visualizer is the administration console for a running Oracle CEP server. For more information, see the Oracle CEP Visualizer User's Guide.

To start the Oracle CEP Visualizer from Oracle CEP IDE for Eclipse:

1. Start the server (see Section 4.3.1, "How to Start an Oracle CEP Server").

2. Right-click the running server in the Servers view and select **Open CEP Visualizer** as shown in Figure 4–21.

Figure 4–21   Opening the Oracle CEP Visualizer
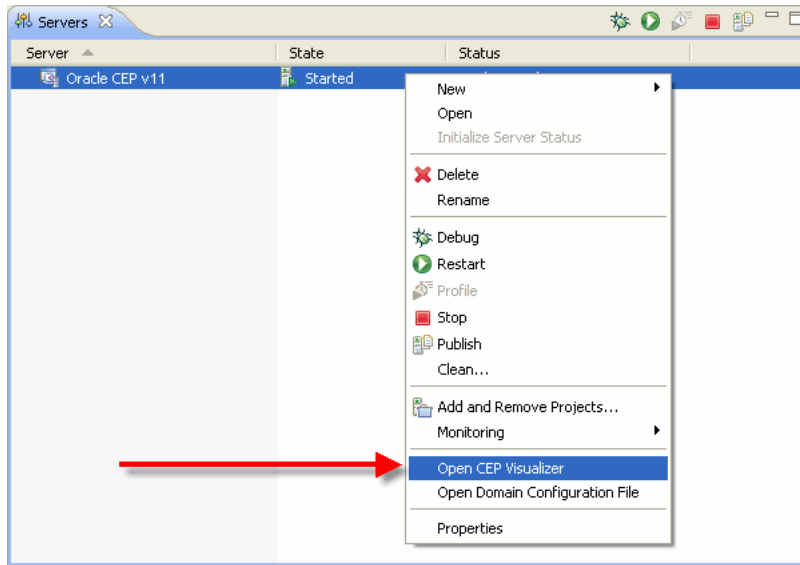
The Oracle CEP Visualizer opens as shown in Figure 4–22.

Figure 4–22   Oracle CEP Visualizer
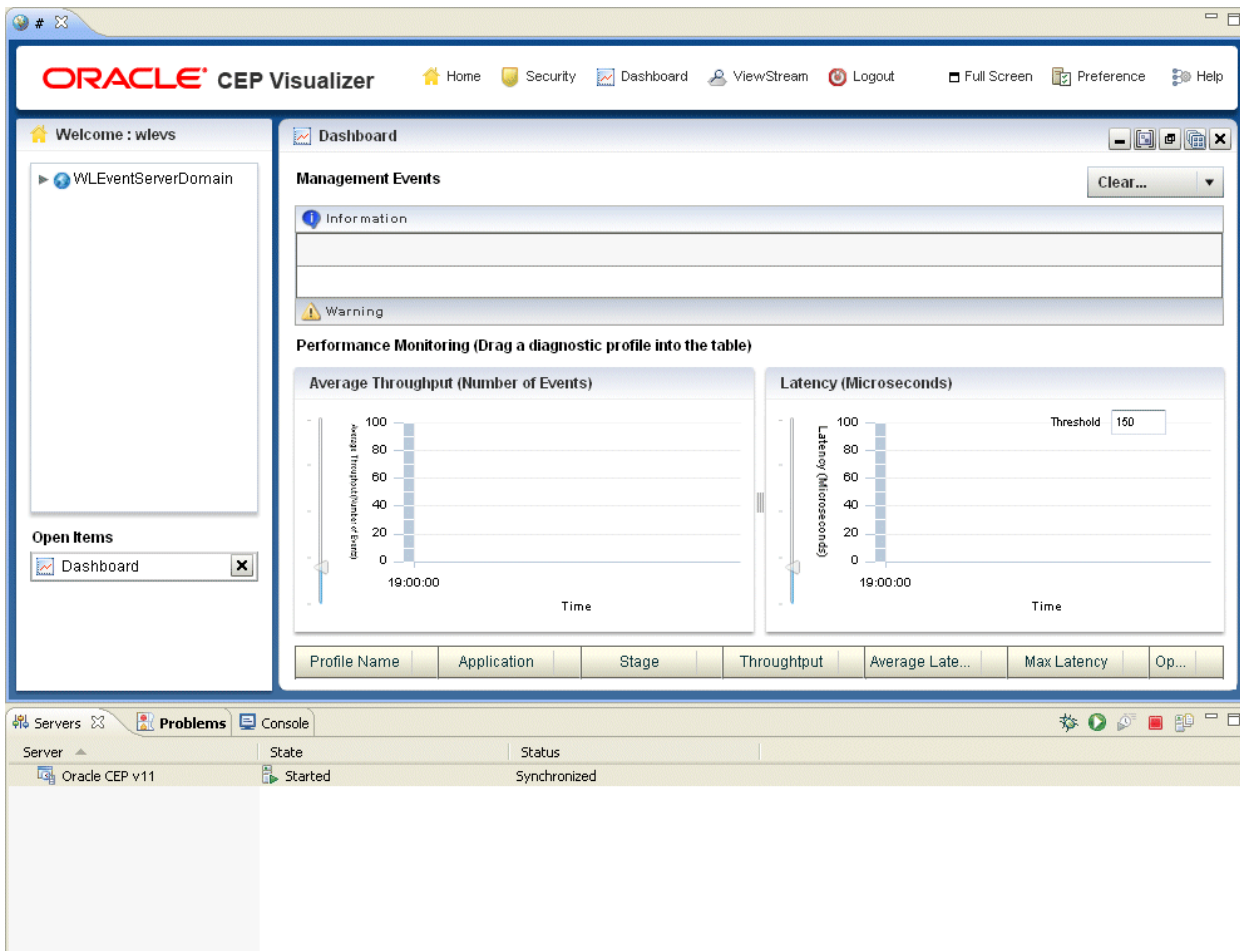
3. Use the Oracle CEP Visualizer as the *Oracle CEP Visualizer User's Guide* describes.

4.4 Debugging an Oracle CEP Application Running on an Oracle CEP Server

Because Oracle CEP applications are java applications, standard Java debugging tools including those provided in Eclipse can be used with these applications.

4.4.1 How to Debug an Oracle CEP Application Running on an Oracle CEP Server

This section describes how to debug an Oracle CEP application running on an Oracle CEP server.

To debug an Oracle CEP application running on an Oracle CEP server:

1. Set a breakpoint in the Java code you wish to debug.

   In this case, set the breakpoint by right-clicking in the gutter of the editor and selecting **Toggle Breakpoint** as Figure 4–23 shows.

Figure 4–23   Setting a Breakpoint
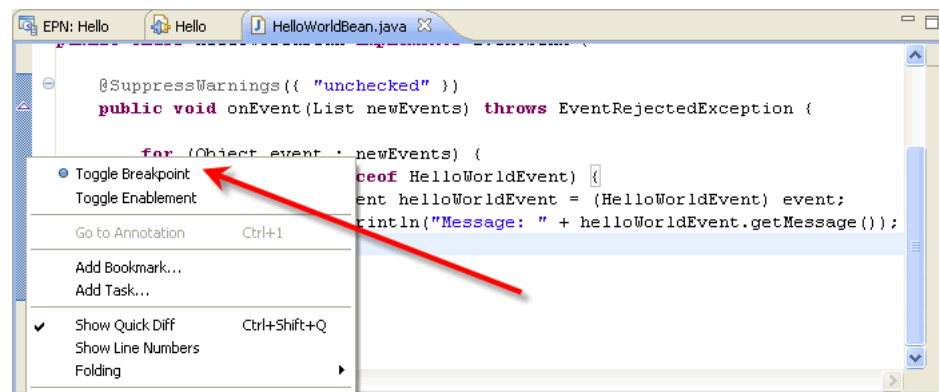
2. Select **Window > Show Views > Servers**.

3. Start the server in debug mode by choosing one of the following as shown in Figure 4–24:

   a. Click the **Start the Server in debug mode** icon in the Servers view tool bar.

   b. Right-click a server in the Servers view and select **Debug**.

Figure 4–24   Starting the Oracle CEP Server in Debug Mode
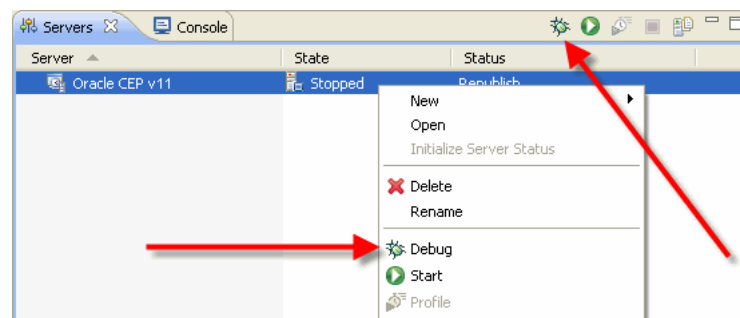
4. The server will start, and when it gets to your breakpoint the thread will stop.

If the Oracle CEP IDE for Eclipse does not automatically switch to the Debug perspective, switch to that perspective by selecting **Window > Open Perspective > Other** and selecting the **Debug** option from the list of perspective.

5. Debug your application using the Debug perspective.

Note: In some cases you may get a dialog box warning that it could not install a breakpoint because of missing line number information. This dialog comes from the core Eclipse debugger and is normally a harmless issue with Oracle CEP Service Engine applications. Simply check the **Don't Tell Me Again** checkbox and continue debugging.

---

6. When you are finished you can stop the server as usual (see Section 4.3.2, "How to Stop an Oracle CEP Server").

5

Oracle CEP IDE for Eclipse and the Event Processing Network

The Oracle CEP Event Processing Network (EPN) is the central concept in an Oracle CEP application. It is the primary point where application components are wired together. Using Oracle CEP IDE for Eclipse, you can use an EPN Editor that provides a graphical view of the EPN and offers visualization and navigation features to help you build Oracle CEP applications.

This section describes how to use the editor and the information it displays, including:

- Section 5.1, "Opening the EPN Editor"

- Section 5.2, "EPN Editor Overview"

- Section 5.3, "Navigating the EPN Editor"

- Section 5.4, "Using the EPN Editor"

5.1 Opening the EPN Editor

You can open the EPN Editor from either the project folder or a context or configuration file of an Oracle CEP application.

5.1.1 How to Open the EPN Editor from a Project Folder

You can open the EPN Editor from the Eclipse project folder of an Oracle CEP application. Alternatively, you can open the EPN Editor from a context or configuration file (see Section 5.1.2, "How to Open the EPN Editor from a Context or Configuration File").

To open the EPN Editor from a project:

1. Launch the Oracle CEP IDE for Eclipse.

2. Open your Oracle CEP project in the Project Explorer.

3. Right-click the project folder and select **Open EPN Editor** as Figure 5–1 shows.

Figure 5–1 Opening the EPN Editor from a Project
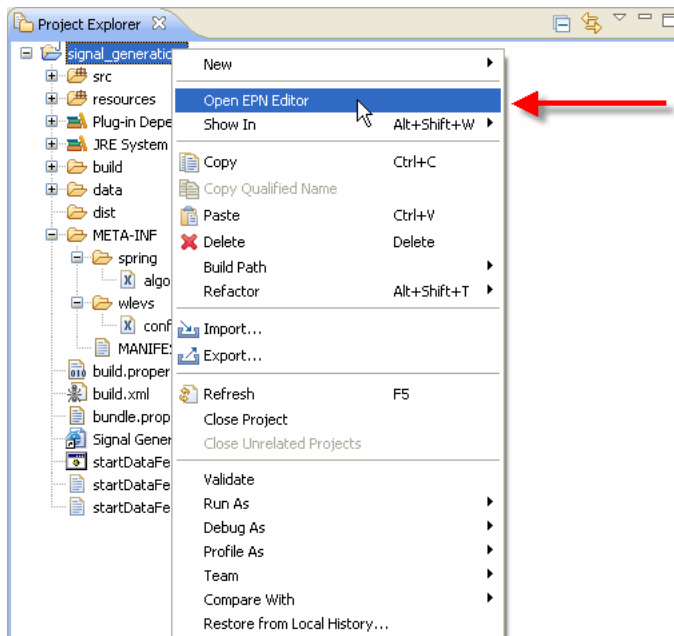
The EPN Editor opens in a tab named $EPN : PROJECT-NAME$, where $PROJECT-NAME$ is the name of your Oracle CEP project, as Figure 5–2 shows.

Figure 5–2 EPN Editor
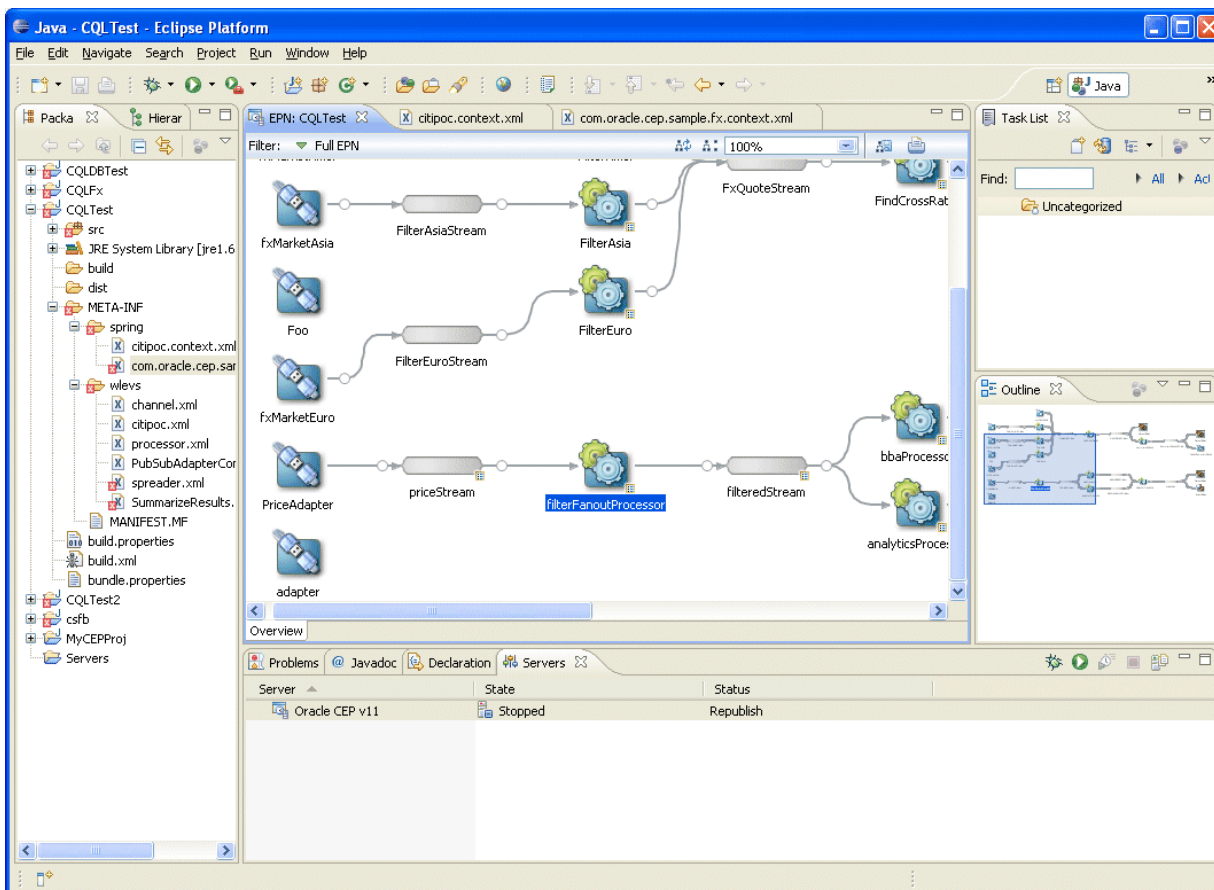
5.1.2 How to Open the EPN Editor from a Context or Configuration File

You can open the EPN Editor from a Spring context file or an Oracle CEP server configuration file in an Oracle CEP application. Alternatively, you can open the EPN Editor from a context or configuration file (see Section 5.1.1, "How to Open the EPN Editor from a Project Folder")

To open the EPN Editor from a context or configuration file:

1. Launch the Oracle CEP IDE for Eclipse.

2. Open your Oracle CEP project in the Project Explorer.

3. Right-click a context or configuration file and select **Open in EPN Editor** as Figure 5–3 shows.

Figure 5–3   Opening the EPN Editor from a Context or Configuration File
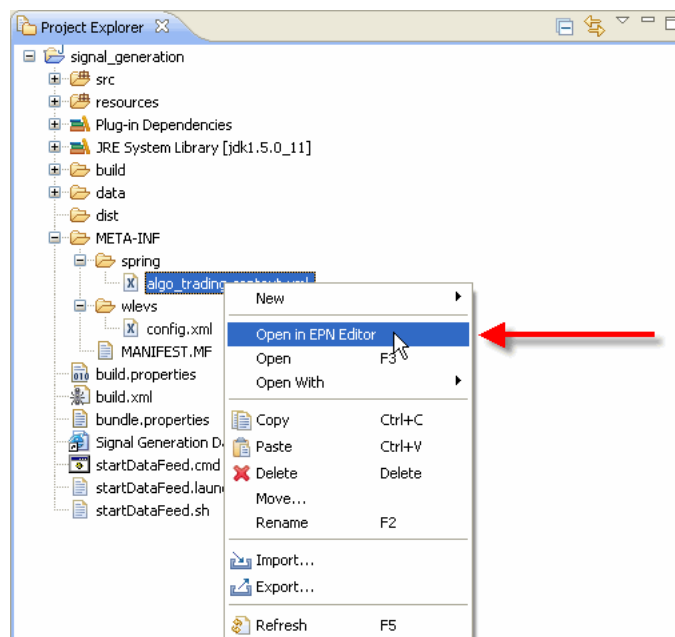
The EPN Editor opens in a tab named `EPN:PROJECT-NAME`, where `PROJECT-NAME` is the name of your Oracle CEP project, as Figure 5–4 shows.

Figure 5–4   EPN Editor
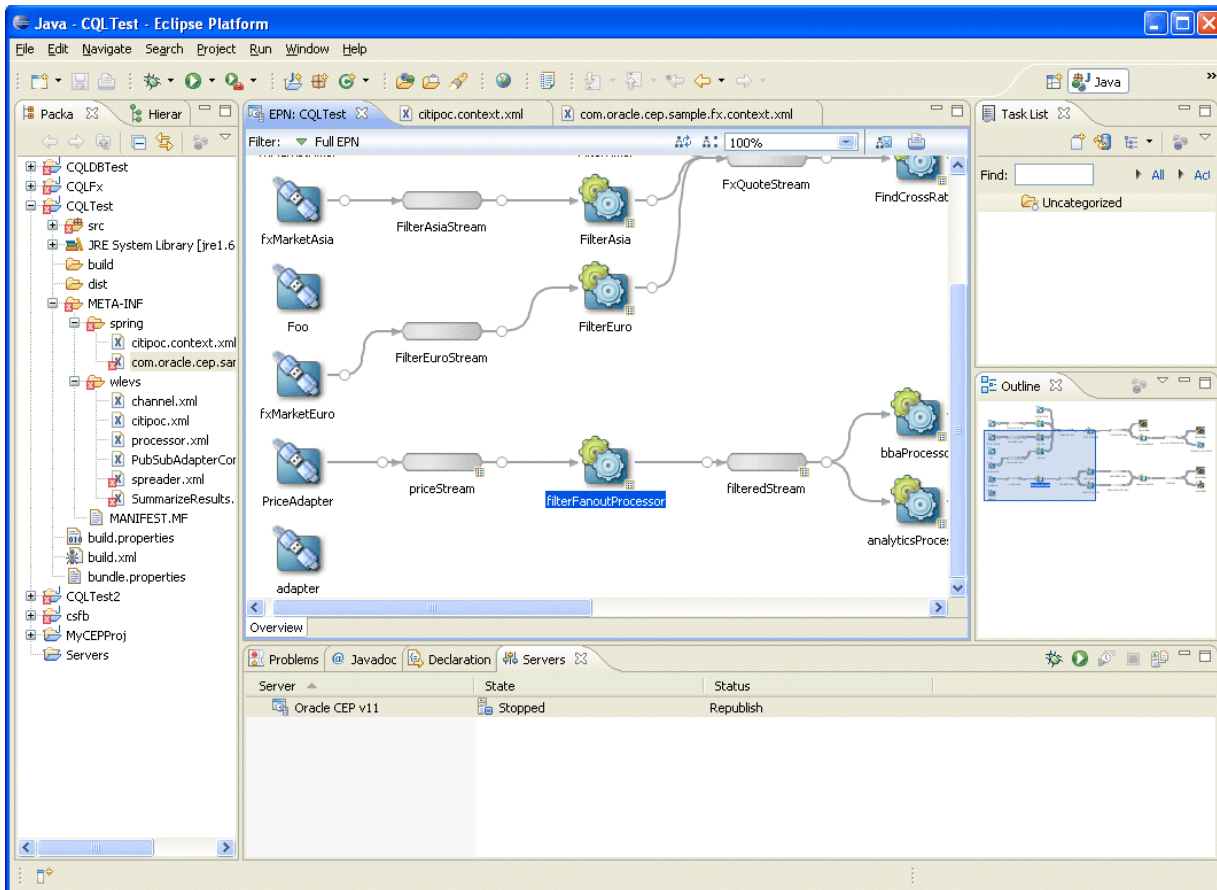
5.2 EPN Editor Overview

This section describes the main controls you use to manage the EPN view and how the EPN Editor displays Oracle CEP application information, including:

- Section 5.2.1, "Flow Representation"

- Section 5.2.2, "Filtering, Zooming, and Layout"

- Section 5.2.3, "Printing and Exporting to an Image"

- Section 5.2.4, "Configuration Badging"

- Section 5.2.5, "Link Specification Location Indicator"

- Section 5.2.6, "Nested Stages"

5.2.1 Flow Representation

The primary display in the editor is of the flow inside the application as Figure 5–5 shows.

Figure 5–5   EPN Flow Representation
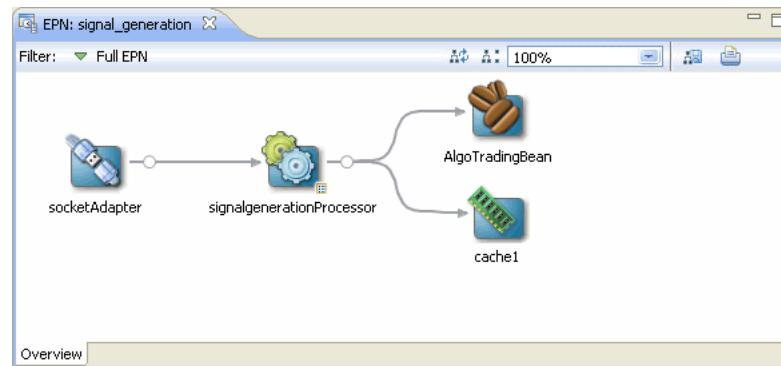
The EPN is composed of nodes connected by links and streams. Nodes are of various types including adapter, processor, database table, bean, and cache. For more information on the graphic notation the EPN Editor uses on nodes, links, and streams, see:

- Section 5.2.4, "Configuration Badging"
- Section 5.2.5, "Link Specification Location Indicator"

5.2.2 Filtering, Zooming, and Layout

Although you often specify your EPN in a single assembly file, you may specify an EPN across multiple assembly files. By default the EPN Editor shows the entire network with the information combined from all files. To see the network for a single assembly file simply select that file from the **Filter** pull-down menu as Figure 5–6 shows.

Figure 5–6   Filtering the EPN by Assembly File
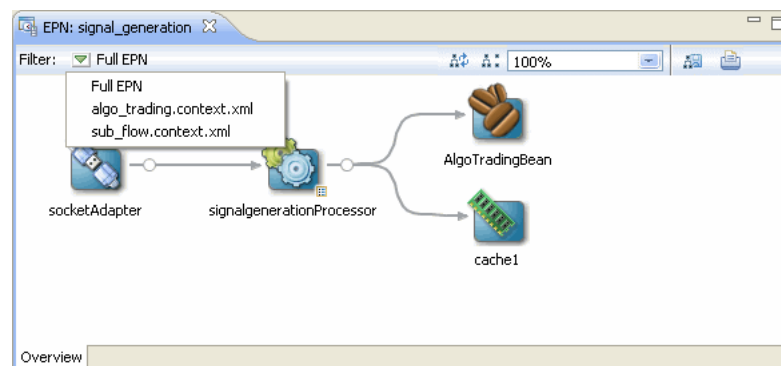
You can optimize and simplify the EPN layout by clicking **Layout EPN** as Figure 5–7 shows.

Figure 5–7   Optimize Layout
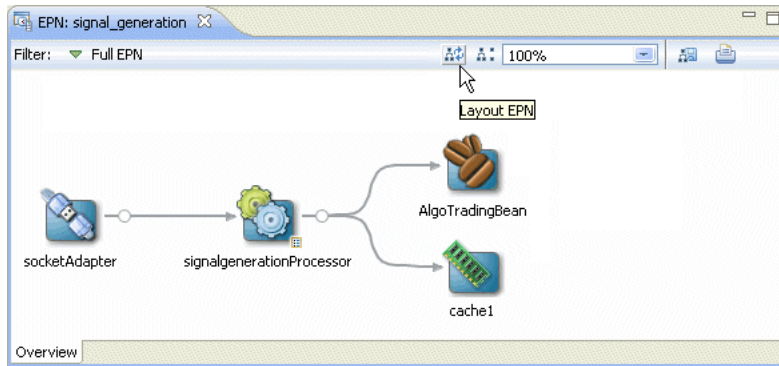
You can also filter out `<bean>` elements with no references in the EPN. Clicking **Show/Hide Unconnected Beans** will toggle the visibility of such beans as Figure 5–8 shows. For more information, see Section 5.4.3, "Laying Out Nodes".

Figure 5–8   Show/Hide Unconnected Beans
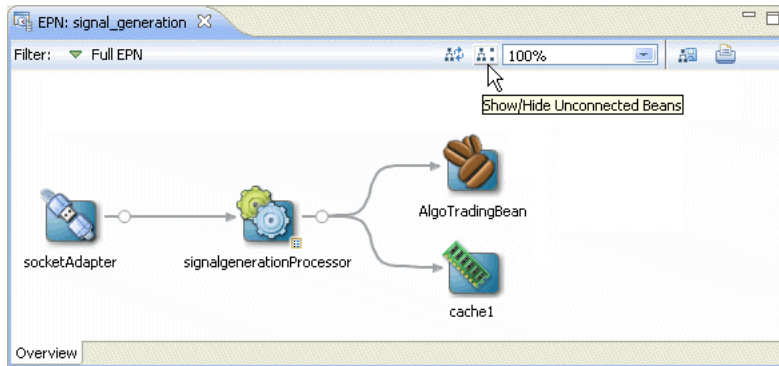
You can change the zoom level of the EPN Editor by entering a percent value into the zoom field or selecting a value from the zoom field pull-down menu as Figure 5–9 shows. To fit the EPN into the current EPN Editor window, select **Fit to Window**.

Figure 5–9   Zoom Level
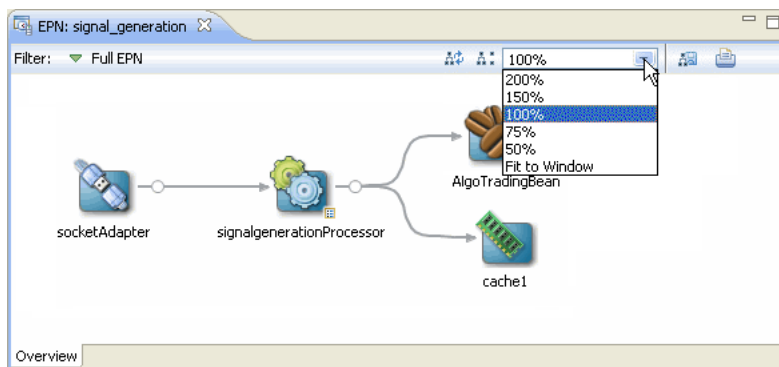
5.2.3 Printing and Exporting to an Image

You can export the EPN Editor view to an image file by clicking **Export to Image** as Figure 5–10 shows. You can export the image as a `.bmp`, `.gif`, `.jpg`, or `.png` file.

Figure 5–10   Exporting the EPN as an Image File
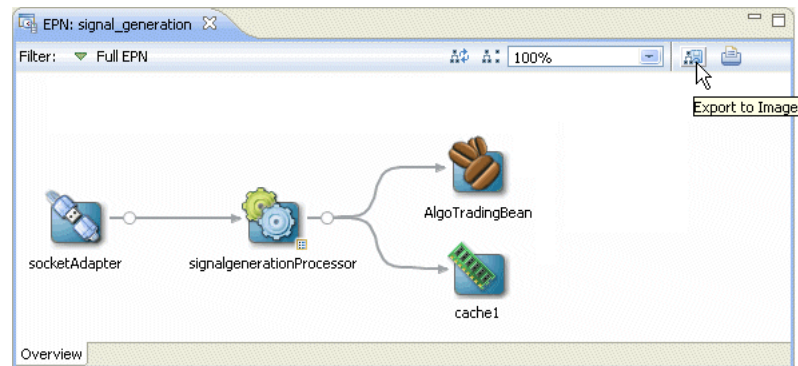
You can print the EPN Editor view by clicking **Print** as Figure 5–11 shows.

Figure 5–11   Printing the EPN
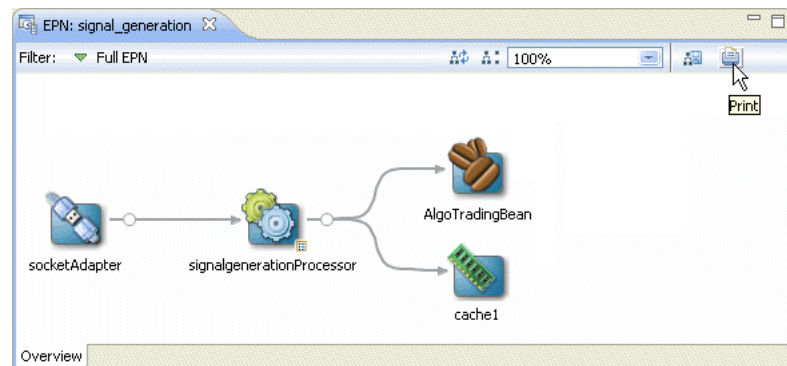
5.2.4 Configuration Badging

Nodes that have configuration information in one of the configuration files in the `META-INF/wlevs` directories are badged with an indicator on the bottom right as Figure 5–12 shows.

Figure 5–12   Configuration Badging

Nodes with this badge will also have the **Go To Configuration Source** context menu item.

5.2.5 Link Specification Location Indicator

When working with streams, you can specify a link in the assembly file as a:

- source element in the downstream node.
- listener element in the upstream node

A circle on the line indicates where a particular link is specified in the assembly file.

Figure 5–13 shows an example in which the link is specified as a source element on the downstream node `outStream` so the circle is next to the `outStream` node. Figure 5–14 shows the corresponding assembly file.

Figure 5–13   Link Source

Figure 5–14   Link Source Assembly File

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

<wlevs:channel id="filteredStream"
    event-type="FilteredPriceEvent">
    <wlevs:listener ref="bbaProcessor" />
    <wlevs:listener ref="analyticsProcessor" />
    <wlevs:source ref="filterFanoutProcessor" />
</wlevs:channel>
```

****

Figure 5–15 shows an example in which the link is specified as a listener element in the upstream node `algoTradingProcessor` so the circle is next to the `algoTradingProcessor` node. Figure 5–16 shows the corresponding assembly file.

Figure 5–15   Link Listener
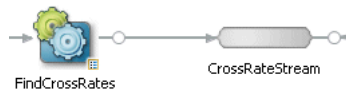
Figure 5–16   Link Listener Assembly File

```
<wlevs:processor id="FindCrossRates" provider="cql">
    <wlevs:listener ref="CrossRateStream"/>
</wlevs:processor>

<wlevs:channel id="CrossRateStream" event-type="SpreaderOuputEvent" advertise="true">
    <wlevs:listener ref="summarizeResults"/>
    <wlevs:listener>
        <bean class="com.oracle.cep.sample.fx.OutputBean" autowire="byName"/>
    </wlevs:listener>
</wlevs:channel>
```

5.2.6 Nested Stages

When you define a child node within a parent node, the child node is said to be nested. Only the parent node can specify the child node as a listener. You can drag references from a nested element, but not to them. For more information, see Section 5.4.2, "Connecting Nodes".

Consider the EPN that Figure 5–17 shows. Example 5–1 shows the EPN assembly source for this EPN. Note that the `HelloWorldBean` is nested within the `helloworldOutputChannel`. As a result, it appears within a box in the EPN diagram. Only the parent `helloworldOutputChannel` may specify the nested bean as a listener.

Figure 5–17   EPN With Nested Bean

Example 5–1   Assembly Source for EPN With Nested Bean

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

<wlevs:channel id="helloworldInputChannel" event-type="HelloWorldEvent" >
    <wlevs:listener ref="helloworldProcessor"/>
    <wlevs:source ref="helloworldAdapter"/>
</wlevs:channel>

<wlevs:processor id="helloworldProcessor" />

<wlevs:channel id="helloworldOutputChannel" event-type="HelloWorldEvent" advertise="true">
    <wlevs:listener>
        <bean class="com.bea.wlevs.example.helloworld.HelloWorldBean"/>
    </wlevs:listener>
    <wlevs:source ref="helloworldProcessor"/>
</wlevs:channel>
```

Alternatively, you can define this EPN so that all nodes are nested as Figure 5–18 shows. Example 5–2 shows the EPN assembly source for this EPN. Note that all the nodes are nested and as a result, all nodes appear within a box in the EPN diagram. The `helloworldAdapter` is the outermost parent node and does not appear within a box in the EPN diagram.

Figure 5–18   EPN With all Nodes Nested

Example 5–2   Assembly Source for EPN With all Nodes Nested

```
<wlevs:adapter id="helloworldAdapter" class="com.bea.wlevs.adapter.example.helloworld.HelloWorldAdapter" >
    <wlevs:instance-property name="message" value="HelloWorld - the current time is:"/>
    <wlevs:listener>
        <wlevs:channel id="helloworldInputChannel" event-type="HelloWorldEvent" >
            <wlevs:listener>
                <wlevs:processor id="helloworldProcessor">
                    <wlevs:listener>
                        <wlevs:channel id="helloworldOutputChannel" event-type="HelloWorldEvent">
                            <wlevs:listener>
                                <bean class="com.bea.wlevs.example.helloworld.HelloWorldBean"/>
                            </wlevs:listener>
                        </wlevs:channel>
                    </wlevs:listener>
                </wlevs:processor>
            </wlevs:listener>
        </wlevs:channel>
    </wlevs:listener>
</wlevs:adapter>
```

5.3 Navigating the EPN Editor

Because the EPN Editor has a view of the whole project it is a natural place from which to navigate to the various artifacts that make up an Oracle CEP application. Oracle CEP IDE for Eclipse offers the following features to help navigate the EPN Editor:

- Section 5.3.1, "Moving the Canvas"
- Section 5.3.2, "Shortcuts to Component Configuration and EPN Assembly Files"
- Section 5.3.3, "Hyperlinking"
- Section 5.3.4, "Context Menus"

5.3.1 Moving the Canvas

To move the EPN canvas without using the horizontal and vertical scroll bars, you can use any of the following options:

- Position the cursor on the canvas, hold down the middle mouse button, and drag.
- Hold down the space bar and click and drag the canvas.
- In the Overview view, click in the highlight box and drag.

5.3.2 Shortcuts to Component Configuration and EPN Assembly Files

If a node has a configuration object associated with it, then double-clicking that node will open the component configuration file where that node's behavior is defined.

Otherwise, double-clicking that node will open the EPN assembly file (the Spring context file) where that node is defined.

A configuration badge will be shown on nodes with associated configuration objects as shown in Figure 5–19.

Figure 5–19   Node with Configuration Badge

exampleProcessorWithConfig

For more information, see:

- Section 5.2.4, "Configuration Badging"
- Section 5.3.3, "Hyperlinking"

5.3.3 Hyperlinking

When editing a component configuration file or EPN assembly file, hold down the **Ctrl** key to turn on hyperlinking. Using hyperlinking, you can easily move between assembly and configuration files and follow reference IDs to jump to bean implementation classes.

Figure 5–20 shows a component configuration file with the cursor over the value of a `processor` element `name` child element while holding down the **Ctrl** key. The `name` value has an underline to indicate it is a hyperlink. Click this link to jump to the corresponding element in the EPN assembly file as Figure 5–21 shows.

Figure 5–20   Component Configuration File: Hyperlinking to EPN Assembly File
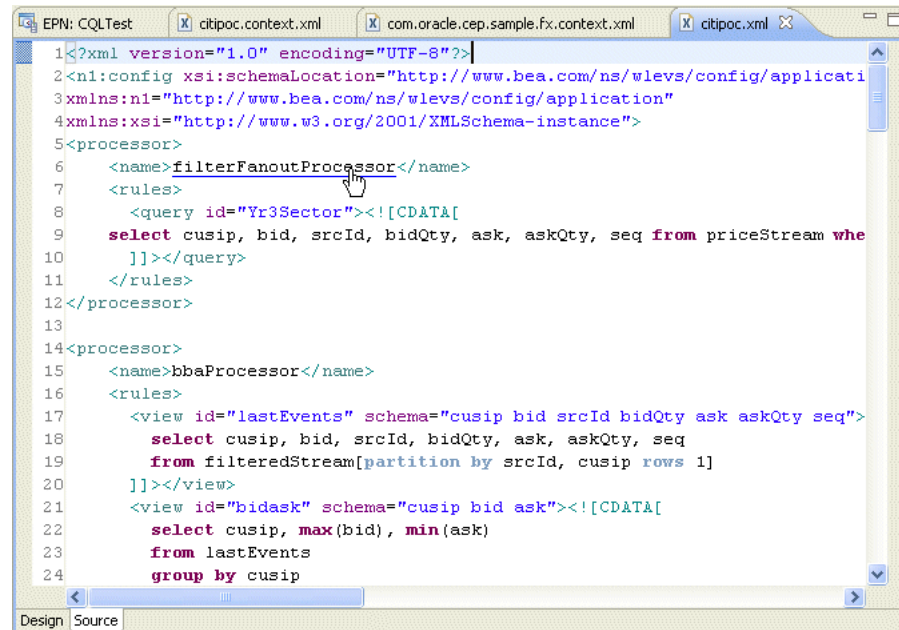
Similarly, hovering over the `wlevs:processor` element `id` child element value `filterFanoutProcessor` while holding down the **Ctrl** key allows you to hyperlink back to the component configuration file.

Figure 5–21   EPN Assembly File: Hyperlinking to Component Configuration File
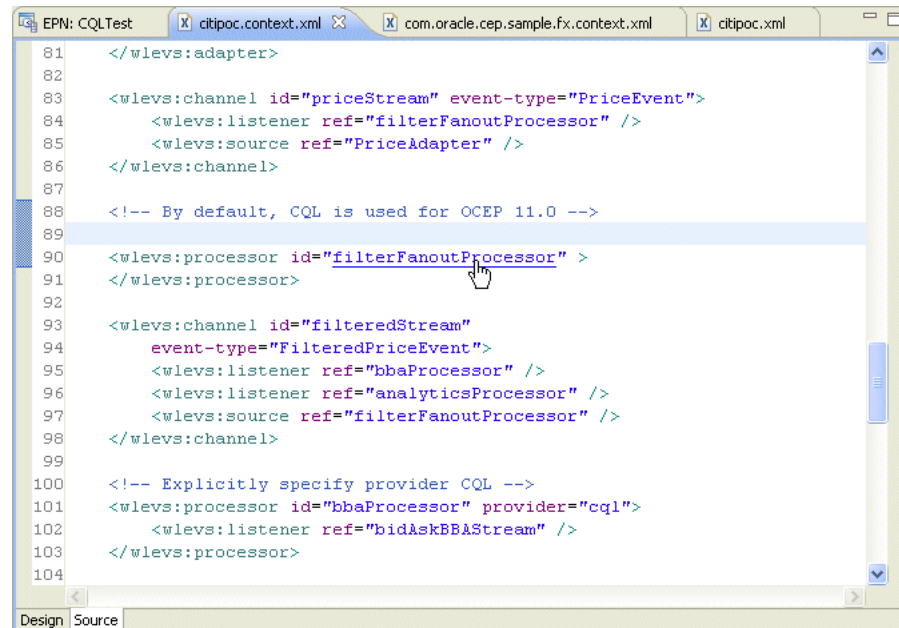
5.3.4 Context Menus

Each node on the EPN Editor has a group of context menu items that provide convenient access to various node-specific functions. Right-click the node to display its context menu.

Depending on the node type, you can use the EPN Editor context menu to select from the following options:

- **Go to Configuration Source**: opens the corresponding component configuration file and positions the cursor in the appropriate element. You can use hyperlinking to quickly move from this file to the corresponding EPN assembly file. For more information, see Section 5.3.3, "Hyperlinking".

- **Go to Assembly Source**: opens the corresponding EPN assembly file and positions the cursor in the appropriate element. You can use hyperlinking to quickly move from this file to the corresponding component configuration file. For more information, see Section 5.3.3, "Hyperlinking"

- **Go to Java Source**: opens the corresponding Java source file for this component.

- **Delete**: deletes the component from both the EPN assembly file and component configuration file (if applicable).

- **Rename**: allows you to change the name of the component. The name is updated in both the EPN assembly file and component configuration file (if applicable).

- **Help**: displays context sensitive help for the component.

Note that these navigation options will become disabled when a corresponding source artifact cannot be found. For example, if an adapter does not have a corresponding entry in a configuration XML file, its **Go to Configuration Source** menu item will be greyed out.

5.4 Using the EPN Editor

The EPN Editor allows you to create and edit an application's EPN using actions on the editor surface. Most actions in the EPN Editor result in edits to an assembly file in that application.

The EPN Editor shows the EPN for a single Oracle CEP application bundle. An application may have more than one assembly file that defines the application EPN as Figure 5–22 shows.

Figure 5–22   EPN Editor Filter
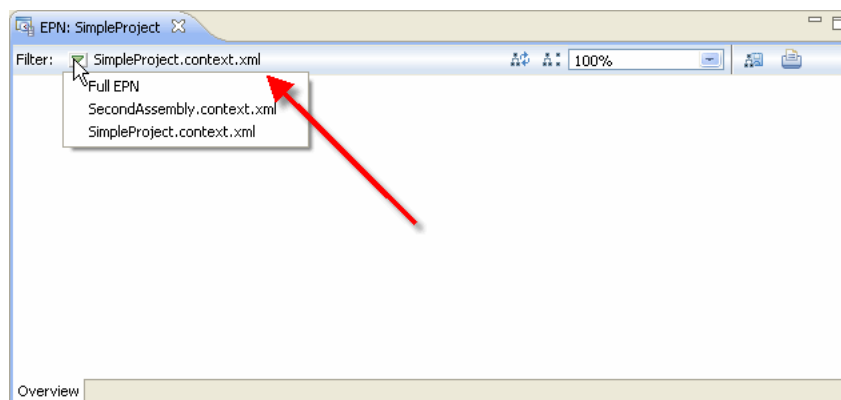
When editing an EPN, the assembly file shown in the EPN Editor filter is the assembly file to which new nodes will be added. If the EPN Editor filter is set to **Full EPN** then the first assembly file in the filter list will be the file to which new nodes will be added. Existing nodes will be edited in or deleted from the assembly file in which they are defined.

If the assembly file the EPN Editor edits is open in an Eclipse source editor, then the edits will be made to the editor for that open file. In this case, you will need to save changes to the open editor before the changes appear in the file on disk.

If the assembly file the EPN Editor edits is not open in an Eclipse source editor, then the edits are immediately applied to the file on disk.

The following sections describe EPN Editor editing tasks, including:

- Section 5.4.1, "Creating Nodes"

- Section 5.4.2, "Connecting Nodes"

- Section 5.4.3, "Laying Out Nodes"

- Section 5.4.4, "Renaming Nodes"

- Section 5.4.5, "Deleting Nodes"

For more information, see:

- Section 3.1, "Oracle CEP Project Overview"

- Section 5.2, "EPN Editor Overview"

- Section 5.1, "Opening the EPN Editor"

- Section 5.3, "Navigating the EPN Editor"

5.4.1 Creating Nodes

When adding new nodes to an EPN using the EPN editor, a new node will appear at the location of the mouse click that was used to show the EPN Editor context menu. You can create any of the nodes that Table 5–1 lists.

Table 5–1 EPN Editor Icons

Node	Description
	Adapter: a node that interfaces an event data source with the EPN or interfaces the EPN with an event data sink.
	For more information, see:
	- Chapter 6, "Configuring JMS Adapters"
	- Chapter 7, "Configuring HTTP Publish-Subscribe Server Adapters"
	- Chapter 8, "Configuring Custom Adapters, Event Beans, and Spring Beans"
	- Chapter 15, "Testing Applications With the Load Generator and csvgen Adapter"
	Bean: a Plain Old Java Object (POJO) node that consumes events. A bean is managed by the Spring framework.
	For more information, see Chapter 8, "Configuring Custom Adapters, Event Beans, and Spring Beans".
	Cache: a node that provides a temporary storage area for events, created exclusively to improve the overall performance of your Oracle CEP application.
	For more information, see Chapter 12, "Configuring Caching".
	Channel: a node that conveys events between an event data source and an event data sink.
	For more information, see Chapter 9, "Configuring Channels".

Table 5–1 (Cont.) EPN Editor Icons

Node	Description
	Event Bean: a node similar to a standard Spring bean except that it can be managed by the Oracle CEP management framework and can actively use the capabilities of the Oracle CEP server container.
	For more information, see Chapter 8, "Configuring Custom Adapters, Event Beans, and Spring Beans".
	Table: a node that connects a relational database table to the EPN as an event data source.
	For more information, see Section 10.3, "Configuring an Oracle CQL Processor Table Source".
	Processor: a node that executes Oracle CQL or EPL rules on the event data offered to it by one or more channels.
	For more information, see:
	■ Chapter 10, "Configuring Oracle CQL Processors"
	■ Chapter 11, "Configuring EPL Processors"

The user may not reposition the nodes on the EPN Editor. To refresh the layout of the nodes on the EPN Editor, click the **Layout EPN** button on the EPN Editor toolbar. For more information, see Section 5.4.3, "Laying Out Nodes".

When a child node is nested within a parent node, its icon appears within a box. For more information, see Section 5.2.6, "Nested Stages".

This section describes:

■ Section 5.4.1.1, "How to Create a Basic Node"

■ Section 5.4.1.2, "How to Create a Processor Node"

5.4.1.1 How to Create a Basic Node

Basic nodes include adapters, beans, caches, channels, event beans, and tables. For information on how to create a processor node, see Section 5.4.1.2, "How to Create a Processor Node".

To create a basic node:

1. Open the EPN Editor (see Section 5.1, "Opening the EPN Editor").

2. Right-click on an empty portion of the EPN Editor surface and select **New** from the context menu as Figure 5–23 shows.

Figure 5–23   Creating a Basic Node
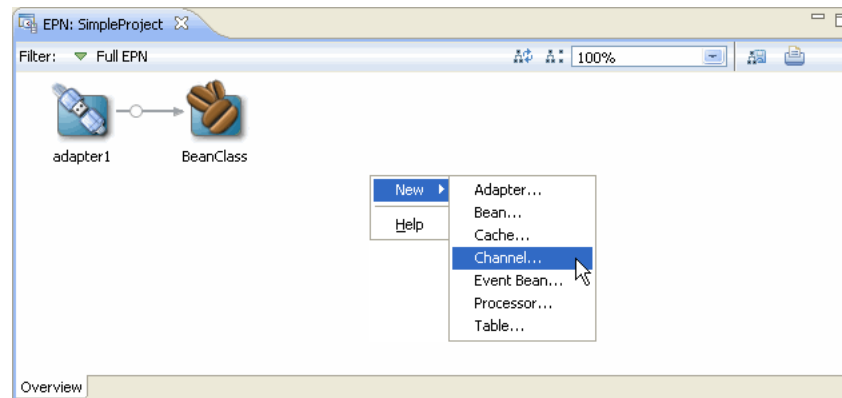
3. Select the type of node you want to create.

    The EPN Editor edits the source file indicated in the EPN Editor filter and the EPN Editor displays the new EPN node. For most nodes, a default ID is chosen and the new node is immediately opened for rename as Figure 5–24 shows.

Figure 5–24   New Basic Node
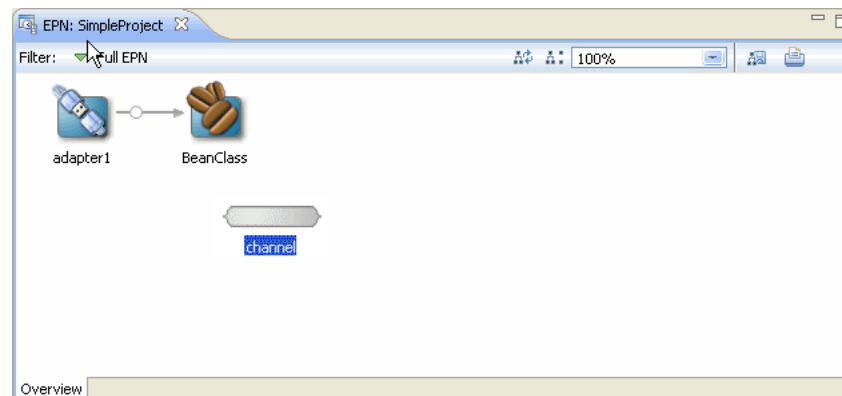
    To rename the node, see Section 5.4.4, "Renaming Nodes".

    To reposition the node and update the EPN Editor layout, see Section 5.4.3, "Laying Out Nodes".

4. Optionally, configure additional node options.

    See:

    ■   Chapter 6, "Configuring JMS Adapters"

    ■   Chapter 7, "Configuring HTTP Publish-Subscribe Server Adapters"

    ■   Chapter 8, "Configuring Custom Adapters, Event Beans, and Spring Beans"

    ■   Chapter 9, "Configuring Channels"

    ■   Section 10.3, "Configuring an Oracle CQL Processor Table Source"

    ■   Chapter 12, "Configuring Caching"

    ■   Chapter 15, "Testing Applications With the Load Generator and csvgen Adapter"

5.4.1.2 How to Create a Processor Node

This section describes how to create a processor node using the EPN Editor. For information on creating other node types, see Section 5.4.1.1, "How to Create a Basic Node".

When deploying an Oracle CEP application with a `wlevs:processor` node, other nodes in an EPN may reference that processor only if a processor configuration exists for that processor. Processor configurations are defined in Oracle CEP application configuration files. See Section 1.1.5, "Component Configuration Files" for more information about CEP configuration files.

To create a processor node:

1. Open the EPN Editor (see Section 5.1, "Opening the EPN Editor").

2. Right-click on an empty portion of the EPN Editor surface and select **New** from the context menu as Figure 5–25 shows.

Figure 5–25   Creating a Processor Node
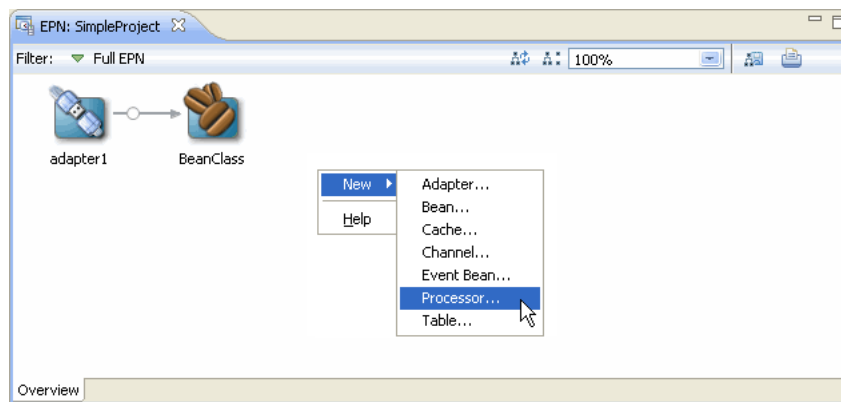
3. Select node type **Processor**.

    The New Processor dialog appears as shown in Figure 5–26.

Figure 5–26   New Processor Dialog
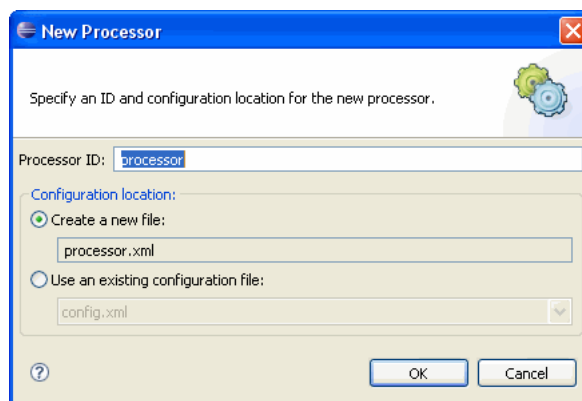
4. Configure the New Processor dialog as shown in Table 5–2.

Table 5–2   New Processor Dialog

Attribute	Description
Processor ID	Specifies the ID of the processor EPN element and the name of the associated processor configuration element
Create a new file	Creates the processor configuration in a new file. The new file is created in the application's `META-INF/wlevs` directory with the same name as the processor ID.
Use an existing configuration file	Creates the processor configuration in an existing configuration file. The new processor configuration element is appended to the configurations in the selected file.

5. Click **OK**.

The EPN Editor creates the processor configuration in the file you specified in the New Processor dialog, edits the source file indicated in the EPN Editor filter, and displays the new EPN node as Figure 5–27 shows.

Figure 5–27   New Processor Node
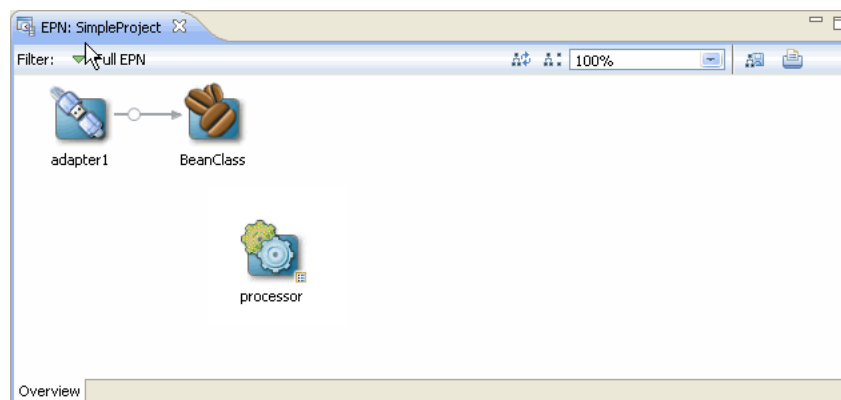
To rename the node, see Section 5.4.4, "Renaming Nodes".

To reposition the node and update the EPN Editor layout, see Section 5.4.3, "Laying Out Nodes".

> **Note:**   In Oracle CEP, you must use a channel to connect an event source to an Oracle CQL processor and to connect an Oracle CQL processor to an event sink. A channel is optional with other Oracle CEP processor types.

6. Optionally, configure additional processor options.

See:

■ Chapter 10, "Configuring Oracle CQL Processors"

■ Chapter 11, "Configuring EPL Processors"

5.4.2 Connecting Nodes

The nodes in the EPN represent the flow of events through an Event Processing Network of an Oracle CEP application. When a node may forward events to another node in the EPN, the EPN Editor allows you to connect that node visually by dragging a line from the source node to the destination node.

5.4.2.1 How to Connect Nodes

This section describes how to connect nodes in the EPN Editor.

To connect nodes:

1. Open the EPN Editor (see Section 5.1, "Opening the EPN Editor").

2. Select the source of events and drag to the target of the event flow.

   - If a connection is allowed, a plug icon is shown at the target end as Figure 5–28 shows.

Figure 5–28 Connecting Nodes: Connection Allowed
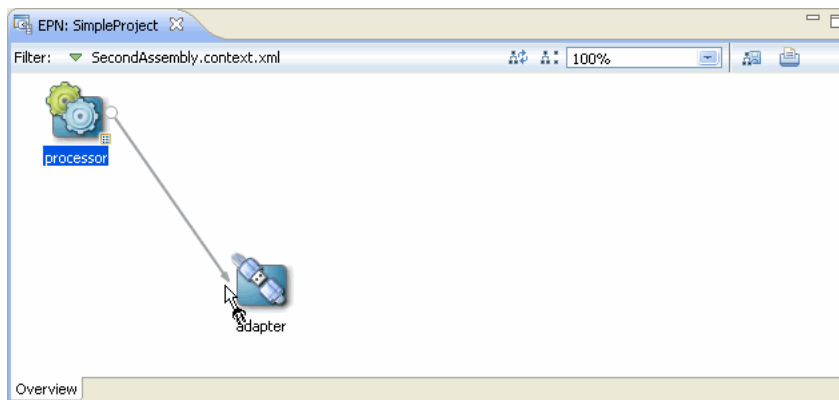
   - If the connection is not allowed, a forbidden icon is shown at the target end as Figure 5–29 shows.

Figure 5–29 Connecting Nodes: Connection Forbidden
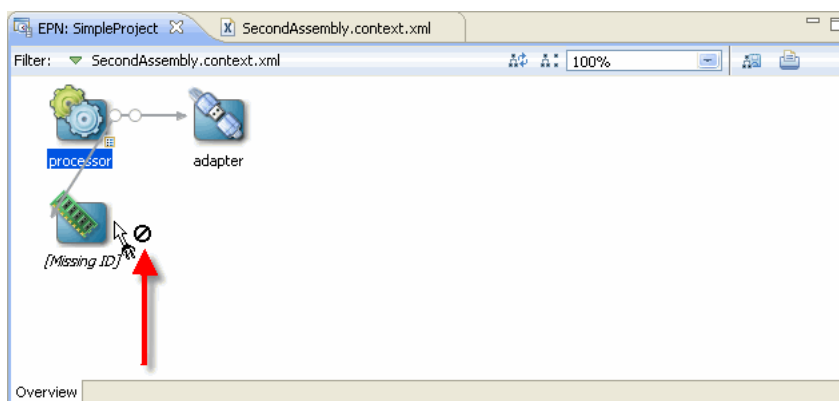
Not all nodes may be a target of event flow. For example, connection is forbidden if:

   – A node does not define a valid identifier.

- A node is nested (for more information, see Section 5.2.6, "Nested Stages").

3. Release the mouse button to complete the connection.

   When the connection is made, the EPN Editor updates the EPN assembly file. For example:

   - When you connect an adapter to a channel or a channel to a processor or event bean, the EPN Editor adds a `wlevs:listener` element to the source node with a reference to the target node by ID.

   - When you connect a table to a processor, the EPN Editor adds a `wlevs:table-source` element to the target processor node that references the source table.

   For example, suppose you connect the adapter to the channel, and the channel to the processor shown in Figure 5–30.

Figure 5–30   Valid Connections

Figure 5–31 shows the EPN assembly file before connection.

Figure 5–31   EPN Assembly File: Before Connection

```
<wlevs:adapter id="adapter" provider="httppub">
</wlevs:adapter>

<wlevs:channel id="channel">
</wlevs:channel>

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

Figure 5–32 shows the EPN assembly file after connection.

Figure 5–32   EPN Assembly File: After Connection

```
<wlevs:adapter id="adapter" provider="httppub">
    <wlevs:listener ref="channel" />
</wlevs:adapter>

<wlevs:channel id="channel">
    <wlevs:listener ref="processor" />
</wlevs:channel>

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

5.4.3 Laying Out Nodes

Certain EPN Editor actions will use the location of the action as the location of the rendered result. For example, when adding new nodes to an EPN using the EPN editor, a new node will appear at the location of the mouse click that was used to show the EPN Editor context menu. The user may not reposition the nodes on the EPN Editor. To refresh the layout of the nodes on the EPN Editor, click the **Layout EPN** button on the EPN Editor toolbar as Figure 5–33 shows.

Figure 5–33   Laying Out Nodes
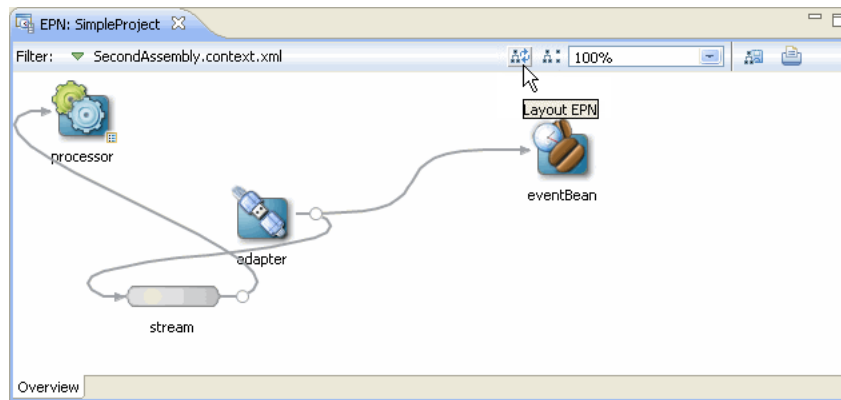
For more information, see Section 5.2.2, "Filtering, Zooming, and Layout".

5.4.4 Renaming Nodes

Most node types support a rename operation that will change all references to the node across both assembly and configuration XML files. You can select **Rename** from the node's context menu as Figure 5–34 shows.

Figure 5–34   Renaming Nodes
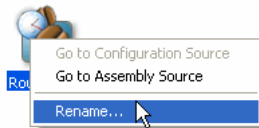
5.4.5 Deleting Nodes

You may delete most nodes and connections visible on the EPN Editor using the node's context menu. Right-click on the object to show the context menu, then select **Delete** as Figure 5–35 shows.

Figure 5–35   Deleting Nodes
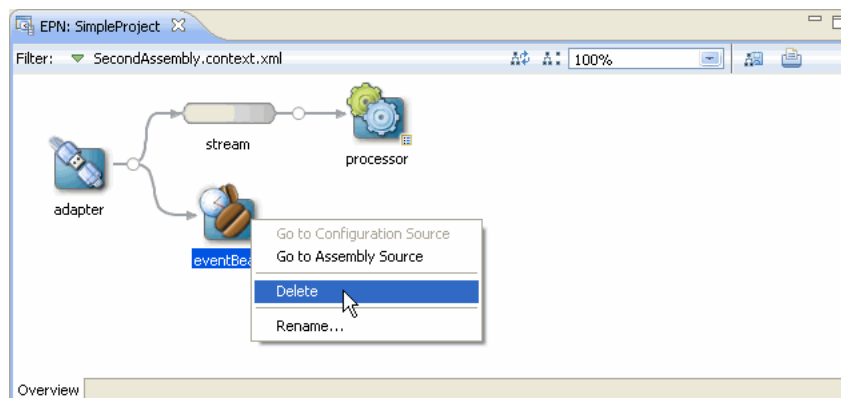
When deleting a node, the incoming and outgoing connections are also deleted. For example, Figure 5–36 shows the EPN and Figure 5–38 shows the assembly file before deleting the channel node named stream.

Figure 5–36   EPN Before Deleting a Channel Node
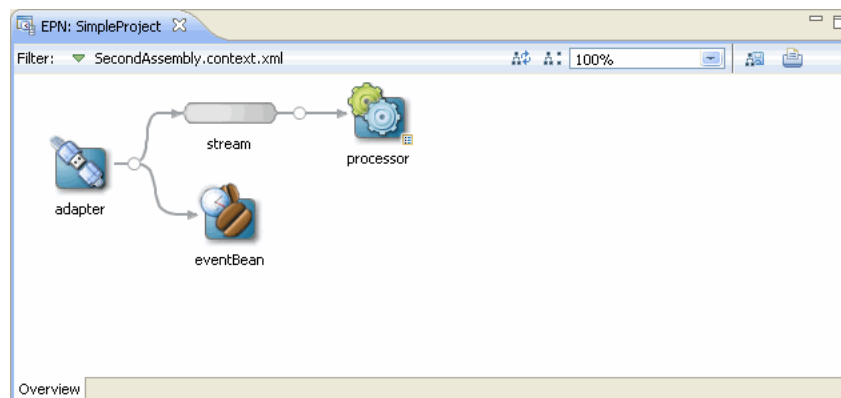
Figure 5–37   Assembly File Before Deleting a Channel Node

```
<wlevs:adapter id="adapter" provider="httppub">
    <wlevs:listener ref="channel" />
</wlevs:adapter>

<wlevs:channel id="channel">
    <wlevs:listener ref="processor" />
</wlevs:channel>

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

Figure 5–38 shows the EPN and Figure 5–39 shows the assembly file after deleting this channel node.

Figure 5–38   EPN After Deleting a Channel Node
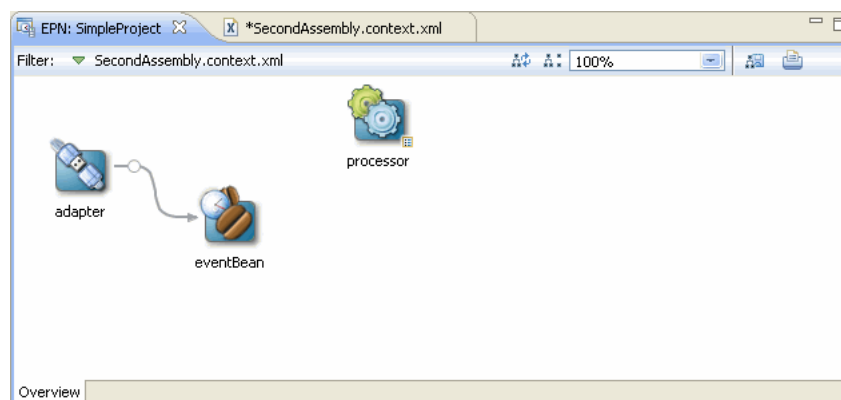
Figure 5–39   Assembly File After Deleting a Channel Node

```
<wlevs:adapter id="adapter" provider="httppub">

</wlevs:adapter>



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

> **Note:** If a `bean` or other anonymous element is deleted, then the object containing that object is deleted too. For example, given a `bean` within a `wlevs:listener` element, then deleting the `bean` will delete the `listener` element too.

Part III

Oracle CEP Event Processing Network

Part III contains the following chapters:

6

Configuring JMS Adapters

This section contains information on the following subjects:

- Section 6.1, "Overview of JMS Adapter Configuration"
- Section 6.2, "Using JMS Adapters"

6.1 Overview of JMS Adapter Configuration

Oracle CEP provides two Java Message Service (JMS) adapters that you can use in your event applications to send and receive messages to and from a JMS queue, respectively, without writing any Java code.

The built-in JMS adapter supports the following modes of operation:

- Section 6.1.1, "Inbound JMS"
- Section 6.1.2, "Outbound JMS"

Oracle CEP supports the following JMS providers:

- Oracle WebLogic JMS
- TIBCO EMS JMS

Oracle CEP includes a WebLogic JMS client. When connecting to Oracle WebLogic server, Oracle CEP uses the T3 client by default. You can use the IIOP WebLogic client by starting Oracle WebLogic Server with the `-useIIOP` command-line argument. This is a server-wide setting that is independent of the JMS code being used (whether it is one of the provided adapters or custom JMS code). It is not possible to mix T3 and IIOP usage within a running Oracle CEP server.

If you are using a JMS provider other than WebLogic JMS, such as TIBCO, you must include the appropriate client jar as a library within your application jar.

For general information about JMS, see Java Message Service on the Sun Developer Network at `http://java.sun.com/products/jms/`.

6.1.1 Inbound JMS

The inbound JMS adapter receives messages from a JMS queue and automatically converts them into events by matching property names with a specified event type. Typically, you also customize this conversion by writing your own Java class to specify exactly how you want the incoming JMS messages to be converted into one or more event types.

If you do not provide your own converter class, and instead let Oracle CEP take care of the conversion between messages and event types, the following is true:

- You must specify an event type that Oracle CEP uses in its conversion. See Section 6.2.3, "Configuring the JMS Adapters" for details.

- By default, the inbound JMS adapter default converter expects incoming JMS messages to be of type `MapMessage`. For each incoming message, an event of the specified event type is created. For each map element in the incoming message, the adapter looks for a property on the event-type and if found, sets the corresponding value.

For more information, see Section 6.1, "Overview of JMS Adapter Configuration".

6.1.2 Outbound JMS

The outbound JMS adapter sends events to a JMS queue, automatically converting the event into a JMS map message by matching property names with the event type. Typically, you also customize this conversion by writing your own Java class to specify exactly how you want the event types to be converted into outgoing JMS messages.

If you do not provide your own converter class, and instead let Oracle CEP take care of the conversion between messages and event types, the following is true:

- You must specify an event type that Oracle CEP uses in its conversion. See Section 6.2.3, "Configuring the JMS Adapters" for details.

- By default, the outbound JMS adapter default converter creates JMS messages of type `MapMessage`. For each property of the event, a corresponding element is created in the output `MapMessage`.

For more information, see Section 6.1, "Overview of JMS Adapter Configuration".

6.2 Using JMS Adapters

The following procedure describes the typical steps to use the JMS adapters provided by Oracle CEP.

> **Note:** It assumed in this section that you have already created an Oracle CEP application along with its EPN assembly file and component configuration files, and that you want to update the application to use an inbound or outbound JMS adapter. If you have not, refer to Chapter 1, "Overview of Creating Oracle CEP Applications" for details.

To use JMS adapters:

1. Optionally create a converter Java class if you want to customize the way JMS messages are converted into event types, or vice versa in the case of the outbound JMS adapter. This step is optional because you can let Oracle CEP make the conversion based on mapping property names between JMS messages and a specified event type.

   See Section 6.2.1, "Creating a Custom Converter Between JMS Messages and Event Types."

2. Update the EPN assembly file of the application by adding a `<wlevs:adapter>` tag for each inbound and outbound JMS adapter you want to use in your application.

   See Section 6.2.2, "Updating the EPN Assembly File With JMS Adapters."

3. Configure the JMS properties of the JMS adapter by updating the component configuration file.

   See Section 6.2.3, "Configuring the JMS Adapters."

4. Update the `MANIFEST.MF` file of your application, adding the package `com.bea.core.encryption` to the `Import-Package` header. For example:

```
Import-Package:
    com.bea.core.encryption
    com.bea.wlevs.adapter.defaultprovider;version="2.0.0.0",
    ...
```

   See Section 14.2.2.1, "Creating the MANIFEST.MF File" for additional information on the manifest file.

6.2.1 Creating a Custom Converter Between JMS Messages and Event Types

If you want to customize the way a JMS message is converted to an event type, or vice versa, you must create your own converter bean.

The custom converter bean for an inbound JMS must implement the `com.bea.wlevs.adapters.jms.api.InboundMessageConverter` interface. This interface has a single method:

```
public List convert(Message message) throws MessageConverterException, JMSException;
```

The `message` parameter corresponds to the incoming JMS message and the return value is a `List` of events that will be passed on to the next stage of the event processing network.

The custom converter bean for an outbound JMS must implement the `com.bea.wlevs.adapters.jms.api.OutboundMessageConverter` interface. This interface has a single method:

```
public List<Message> convert(Session session, Object event) throws MessageConverterException,
JMSException;
```

The parameters correspond to an event received by the outbound JMS adapter from the source node in the EPN and the return value is a `List` of JMS messages.

See the *Oracle CEP Java API Reference* for a full description of these APIs.

The following example shows the Java source of a custom converter bean that implements both `InboundMessageConverter` and `OutboundMessageConvert`; this bean can be used for both inbound and outbound JMS adapters:

```
package com.customer;
import com.bea.wlevs.adapters.jms.api.InboundMessageConverter;
import com.bea.wlevs.adapters.jms.api.MessageConverterException;
import com.bea.wlevs.adapters.jms.api.OutboundMessageConverter;
import javax.jms.JMSException;
import javax.jms.Message;
import javax.jms.Session;
import javax.jms.TextMessage;
import java.util.ArrayList;
import java.util.List;
public class MessageConverter implements InboundMessageConverter, OutboundMessageConverter {
    public List convert(Message message) throws MessageConverterException, JMSException {
        TestEvent event = new TestEvent();
        TextMessage textMessage = (TextMessage) message;
        event.setString_1(textMessage.getText());
        List events = new ArrayList(1);
        events.add(event);
```

```
            return events;
        }
      public List<Message> convert(Session session, Object inputEvent) throws
MessageConverterException, JMSException {
            TestEvent event = (TestEvent) inputEvent;
            TextMessage message = session.createTextMessage(
                "Text message: " + event.getString_1()
            );
            List<Message> messages = new ArrayList<Message>();
            messages.add(message);
            return messages;
        }
}
```

6.2.2 Updating the EPN Assembly File With JMS Adapters

For each JMS adapter in your event processing network, you must add a corresponding `<wlevs:adapter>` tag to the EPN assembly file of your application; use the `provider` attribute to specify whether the JMS adapter is inbound or outbound. Follow these guidelines:

- If you are specifying an inbound JMS adapter, set the `provider` attribute to `jms-inbound`, as shown:

  ```
  <wlevs:adapter id="jmsInbound" provider="jms-inbound"/>
  ```

  The value of the `id` attribute, in this case `jmsInbound`, must match the name specified for this JMS adapter in its configuration file. The configuration file configures the JMS queue from which this inbound JMS adapter gets its messages.

  Because no converter bean is specified, Oracle CEP automatically converts the inbound message to the event type specified in the component configuration file by mapping property names.

- If you are specifying an outbound JMS adapter, set the `provider` attribute to `jms-outbound`, as shown:

  ```
  <wlevs:adapter id="jmsOutbound" provider="jms-outbound"/>
  ```

  The value of the `id` attribute, in this case `jmsOutbound`, must match the name specified for this JMS adapter in its configuration file. The configuration file configures the JMS queue to which this outbound JMS adapter sends messages.

  Because no converter bean is specified, Oracle CEP automatically converts the incoming event types to outgoing JMS messages by mapping the property names.

- For both inbound and outbound JMS adapters, if you have created a custom converter bean to customize the conversion between the JMS messages and event types, first use the standard `<bean>` Spring tag to declare it in the EPN assembly file. Then pass a reference of the bean to the JMS adapter by specifying its `id` using the `<wlevs:instance-property>` tag, with the `name` attribute set to `converterBean`, as shown:

  ```
  <bean id="myConverter"
        class="com.customer.MessageConverter"/>
  <wlevs:adapter id="jmsOutbound" provider="jms-outbound">
      <wlevs:instance-property name="converterBean" ref="myConverter"/>
  </wlevs:adapter>
  ```

  In this case, be sure you do *not* specify an event type in the component configuration file because it is assumed that the custom converter bean takes care of specifying the event type.

As with any other stage in the EPN, add listeners to the `<wlevs:adapter>` tag to integrate the JMS adapter into the event processing network. Typically, an inbound JMS adapter is the first stage in an EPN (because it receives messages) and an outbound JMS adapter would be in a later stage (because it sends messages). However, the requirements of your own Oracle CEP application define where in the network the JMS adapters fit in.

The following sample EPN assembly file shows how to configure an outbound JMS adapter. The network is simple: a custom adapter called `getData` receives data from some feed, converts it into an event type and passes it to `myProcessor`, which in turn sends the events to the `jmsOutbound` JMS adapter via the `streamOne` channel. Oracle CEP automatically converts these events to JMS messages and sends the messages to the JMS queue configured in the component configuration file associated with the `jmsOutbound` adapter.

```xml
<?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-v11_0_0.xsd">
    <wlevs:event-type-repository>
        <wlevs:event-type type-name="JMSEvent">
            <wlevs:class>com.customer.JMSEvent</wlevs:class>
        </wlevs:event-type>
    </wlevs:event-type-repository>
    <!-- Custom adapter that gets data from somewhere and sends it to myProcessor -->
    <wlevs:adapter id="getData"
                   class="com.customer.GetData">
        <wlevs:listener ref="myProcessor"/>
    </wlevs:adapter>
    <wlevs:processor id="myProcessor" />
    <wlevs:adapter id="jmsOutbound" provider="jms-outbound"/>
     <!-- Channel for events flowing from myProcessor to outbound JMS adapter -->
    <wlevs:channel id="streamOne">
       <wlevs:listener ref="jmsOutbound"/>
       <wlevs:source ref="myProcessor"/>
    </wlevs:channel>
</beans>
```

The following sample EPN assembly file shows how to configure an inbound JMS adapter. The network is simple: the inbound JMS adapter called `jmsInbound` receives messages from the JMS queue configured in its component configuration file. The Spring bean `myConverter` converts the incoming JMS messages into event types, and then these events flow to the `mySink` event bean.

```xml
<?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-v11_0_0.xsd">
```

```
                    <wlevs:event-type-repository>
                        <wlevs:event-type type-name="JMSEvent">
                            <wlevs:class>com.customer.JMSEvent</wlevs:class>
                        </wlevs:event-type>
                    </wlevs:event-type-repository>
                    <!-- Event bean that is an event sink -->
                    <wlevs:event-bean id="mySink"
                                      class="com.customer.MySink"/>
                    <!-- Inbound JMS adapter with custom converter class; adapter sends events to mySink
event bean-->
                  <bean id="myConverter" class="com.customer.MessageConverter"/>
                  <wlevs:adapter id="jmsInbound" provider="jms-inbound">
                      <wlevs:instance-property name="converterBean" ref="myConverter"/>
                      <wlevs:listener ref="mySink"/>
                  </wlevs:adapter>
              </beans>
```

6.2.3 Configuring the JMS Adapters

You configure the JMS adapters in their respective configuration files, similar to how you configure other components in the event processing network, such as processors or streams. For general information about these configuration files, see Section 1.1.5, "Component Configuration Files."

The root element for configuring a JMS adapter is `jms-adapter`. The `name` child element for a particular adapter must match the `id` attribute of the corresponding `wlevs:adapter` element in the EPN assembly file that declares this adapter.

The following table describes the additional child elements of `jms-adapter` you can configure for both inbound and outbound JMS adapters.

Table 6–1   Child Elements of jms-adapter for Inbound and Outbound Adapters

Child Element	Description
event-type	Event type whose properties match the JMS message properties.
	Specify this child element *only* if you want Oracle CEP to automatically perform the conversion between JMS messages and events. If you have created your own custom converter bean, then do not specify this element.
jndi-provider-url	Required. The URL of the JNDI provider.
jndi-factory	Optional. The JNDI factory name. Default value is `weblogic.jndi.WLInitialContextFactory,` for Oracle CEP server JMS.
connection-jndi-name	Optional. The JNDI name of the JMS connection factory. Default value is `weblogic.jms.ConnectionFactory,` for Oracle CEP server JMS.
destination-jndi-name destination-name	Required. Either the JNDI name, or actual name, of the JMS destination. Specify one or the other, but not both.
user	Optional. The username for the external resource.
password	Optional The password for the external resource.
encrypted-password	Optional. The encrypted password for the external resource.
	See Section 6.2.4, "Encrypting Passwords in the JMS Adapter Configuration File" for details on encrypting the password.

The following table lists the optional child elements of `jms-adapter` you can configure for inbound JMS adapters only.

Table 6–2   Optional Child Elements for Inbound JMS Adapters

Child Element	Description
work-manager	Name of a work manager, configured in the server's `config.xml` file. This name corresponds to the value of the `<name>` child element of the `<work-manager>` element in `config.xml`. The default value is the work manager configured for the application itself.
concurrent-consumers	Number of consumers to create. Default value is 1. If you set this value to number greater than one, be sure that your converter bean is thread-safe. This is because the converter bean will be shared among the consumers.
message-selector	JMS message selector to use to filter messages.
session-ack-mode-name	Session acknowledgement mode.
session-transacted	Boolean value that specifies whether to use transacted sessions.

The following table lists the optional child elements of `jms-adapter` you can configure for outbound JMS adapters only.

Table 6–3   Optional Child Elements for Outbound JMS Adapters

Child Element	Description
delivery-mode	Specifies the delivery mode: `persistent` (default value) or `nonpersistent`.

For the full schema for the configuration of JMS adapters, see Section B.1, "Component Configuration Schema wlevs_application_config.xsd".

The following configuration file shows a complete example of configuring both an inbound and outbound JMS adapter.

```xml
<?xml version="1.0" encoding="UTF-8"?>
<n1:config
 xsi:schemaLocation="http://www.bea.com/ns/wlevs/config/application wlevs_application_
config.xsd"
 xmlns:n1="http://www.bea.com/ns/wlevs/config/application"
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
    <jms-adapter>
        <name>jmsInbound</name>
        <jndi-provider-url>t3://localhost:7001</jndi-provider-url>
        <destination-jndi-name>Queue1</destination-jndi-name>
        <user>weblogic</user>
        <password>weblogic</password>
        <work-manager>JettyWorkManager</work-manager>
        <concurrent-consumers>1</concurrent-consumers>
        <session-transacted>false</session-transacted>
    </jms-adapter>
    <jms-adapter>
        <name>jmsOutbound</name>
        <event-type>JMSEvent</event-type>
        <jndi-provider-url>t3://localhost:7001</jndi-provider-url>
        <destination-jndi-name>Topic1</destination-jndi-name>
        <delivery-mode>nonpersistent</delivery-mode>
    </jms-adapter>
</n1:config>
```

The following snippet shows how to configure an inbound JMS adapter connecting to TIBCO EMS JMS:

```xml
<jms-adapter>
  <name>myJmsAdapter</name>
  <jndi-provider-url>t3://localhost:7222</jndi-provider-url>
  <jndi-factory>com.tibco.tibjms.naming.TibjmsInitialContextFactory</jndi-factory>
```

```
      <connection-jndi-name>TibcoQueueConnectionFactory</connection-jndi-name>
      <destination-jndi-name>MyQueue</destination-jndi-name>
</jms-adapter>
```

6.2.4 Encrypting Passwords in the JMS Adapter Configuration File

You can encrypt the password in the JMS adapter configuration file.

> **Note:** The procedure assumes that you are currently using the
> `<password>` element in the configuration file, along with a cleartext
> password value, but want to start using the
> `<encrypted-password>` element to encrypt the password.

To encrypt passwords in the JMS adapter configuration file:

1. Open a command window and set your environment as described in "Setting Your
    Development Environment" in the *Oracle CEP Getting Started*.

2. Change to the directory that contains the configuration file for your JMS adapter.

3. Execute the following `encryptMSAConfig` command to encrypt the value of the
    `<password>` element in the configuration file:

    ```
    prompt> ORACLE_CEP_HOME/ocep_11.1/bin/encryptMSAConfig . config_file
    msainternal.dat_file
    ```

    where *ORACLE_CEP_HOME* refers to the main BEA directory into which you
    installed Oracle CEP, such as `d:\oracle_cep`. The second argument refers to
    the directory that contains the JMS adapter configuration file; because this
    procedure directs you to actually change to the directory, the example shows `"."`.
    The *config_file* parameter refers to the name of your JMS adapter
    configuration file. Finally, the *msainternal.dat_file* parameter refers to the
    location of the `.msainternal.dat` file associated with your domain; by default
    this file is located in the *DOMAIN_DIR*/*servername* directory, where *DOMAIN_
    DIR* refers to the domain directory such as `/oracle_cep/user_
    projects/domains/mydomain` and *servername* refers to the server instance.

    The `encryptMSAConfig` command comes in two flavors:
    `encryptMSAConfig.cmd` (Windows) and `encryptMSAConfig.sh` (UNIX).

    After you run the command, the value of the `<password>` element will be
    encrypted, as shown in bold in the following example:

    ```
    <jms-adapter>
      <name>jmsInbound</name>
      <jndi-provider-url>t3://localhost:7001</jndi-provider-url>
      <destination-jndi-name>Queue1</destination-jndi-name>
      <user>weblogic</user>
      <password>{Salted-3DES}B7L6nehu7dgPtJJTnTJWRA==</password>
      <work-manager>JettyWorkManager</work-manager>
      <concurrent-consumers>1</concurrent-consumers>
      <session-transacted>false</session-transacted>
    </jms-adapter>
    ```

4. Using your favorite XML editor, edit the JMS adapter configuration file. Change
    the `<password>` element (whose value is now encrypted) to
    `<encrypted-password>`, as shown in bold in the following example:

    ```
    <jms-adapter>
      <name>jmsInbound</name>
    ```

```
<jndi-provider-url>t3://localhost:7001</jndi-provider-url>
<destination-jndi-name>Queue1</destination-jndi-name>
<user>weblogic</user>
<encrypted-password>{Salted-3DES}B7L6nehu7dgPtJJTnTJWRA==</encrypted-password
>
<work-manager>JettyWorkManager</work-manager>
<concurrent-consumers>1</concurrent-consumers>
<session-transacted>false</session-transacted>
</jms-adapter>
```

7

Configuring HTTP Publish-Subscribe Server Adapters

This section contains information on the following subjects:

- Section 7.1, "Overview of HTTP Publish-Subscribe Server Adapter Configuration"
- Section 7.2, "Using the Built-In HTTP Pub-Sub Adapters in an Application"

7.1 Overview of HTTP Publish-Subscribe Server Adapter Configuration

An HTTP Publish-Subscribe server (pub-sub server) is a mechanism whereby Web clients, such as browser-based clients, subscribe to channels, receive messages as they become available, and publish messages to these channels, all using asynchronous messages over HTTP. A channel is similar to a JMS topic.

Every instance of Oracle CEP includes a pub-sub server that programmers can use to implement HTTP publish-subscribe functionality in their applications. The pub-sub server is configured in the `config.xml` file along with other server services such as Jetty and JDBC datasources. The pub-sub server is based on the Bayeux protocol (see `http://svn.xantus.org/shortbus/trunk/bayeux/bayeux.html`) proposed by the cometd project (see `http://cometd.com/`). The Bayeux protocol defines a contract between the client and the server for communicating with asynchronous messages over HTTP.

In Oracle CEP, programmers access HTTP publish-subscribe functionality by using the following built-in HTTP publish-subscribe adapters (pub-sub adapters):

- Publishing to a channel: see Section 7.1.1, "Overview of the Built-In Pub-Sub Adapter for Publishing"
  - Local publishing to a channel: see Section 7.1.1.1, "Local Publishing".
  - Remote publishing to a channel: see Section 7.1.1.2, "Remote Publishing".
- Subscribing to a channel: see Section 7.1.2, "Overview of the Built-In Pub-Sub Adapter for Subscribing".

Oracle CEP also provides a pub-sub API for programmers to create their own custom pub-sub adapters for publishing and subscribing to a channel, if the built-in pub-sub adapters are not adequate. For example, programmers might want to filter incoming messages from a subscribed channel, dynamically create or destroy local channels, and so on. The built-in pub-sub adapters do not provide this functionality, which is why programmers must implement their own custom pub-sub adapters in this case. For details, see Chapter 8, "Configuring Custom Adapters, Event Beans, and Spring Beans."

By default, Oracle CEP performs automatic conversion to and from Oracle CEP event types. Alternatively, you can create a custom converter. See Section 7.2.1, "Creating a Custom Converter Between the HTTP Pub-Sub Messages and Event Types".

Oracle CEP can also automatically convert between JSON messages and Oracle CEP event types. See Section 7.1.3, "Converting Between JSON Messages and Event Types".

The built-in pub-sub adapters work like any other adapter: they are stages in the event processing network, they are defined in the EPN assembly file, and they are configured with the standard component configuration files. Typical configuration options include specifying channels, specifying the local or remote pub-sub server, and user authentication.

The pub-sub server can communicate with any client that can understand the Bayeux protocol. Programmers developer their Web clients using one of the following frameworks:

- Dojo JavaScript library (see `http://dojotoolkit.org/`) that supports the Bayeux protocol. Oracle CEP does not provide this library.

- WebLogic Workshop Flex plug-in that enables development of a Flex client that uses the Bayeux protocol to communicate with a pub-sub server.

For information on securing an HTTP pub-sub server channel, see "Configuring HTTP Publish-Subscribe Server Channel Security" in the *Oracle CEP Administrator's Guide*.

7.1.1 Overview of the Built-In Pub-Sub Adapter for Publishing

You can use the built-in pub-sub adapter for publishing events to a channel. The built-in pub-sub adapter supports the following publishing modes:

- Section 7.1.1.1, "Local Publishing"

- Section 7.1.1.2, "Remote Publishing"

For more information, see Section 7.1, "Overview of HTTP Publish-Subscribe Server Adapter Configuration".

7.1.1.1 Local Publishing

Figure 7–1 shows how the built-in pub-sub adapter for local publishing fits into a simple event processing network. The arbitrary adapter and processor are not required, they are just an example of possible components in your application in addition to the pub-sub adapter.

Figure 7–1   Built-In Pub-Sub Adapter For Local Publishing
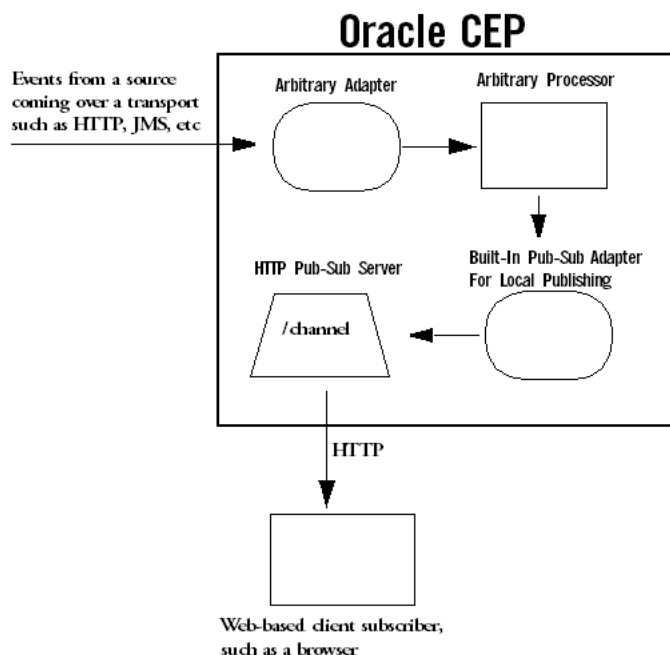
Note the following in Figure 7–1:

- Events flow from some source into an adapter of an application running in Oracle CEP. This adapter is not required, it is shown only as an example.

- The events flow from the adapter to an arbitrary processor; again, this processor is not required.

- The processor sends the events to the built-in pub-sub adapter for local publishing. The adapter in turn sends the events to the local HTTP pub-sub server configured for the Oracle CEP instance on which the application is deployed. The pub-sub adapter sends the messages to the channel for which it has been configured.

- The local HTTP pub-sub server configured for Oracle CEP then sends the event as a message to all subscribers of the local channel.

7.1.1.2 Remote Publishing

Figure 7–2 shows how the built-in pub-sub adapter for remote publishing fits into a simple event processing network.

Figure 7–2   Built-In Pub-Sub Adapter For Remote Publishing
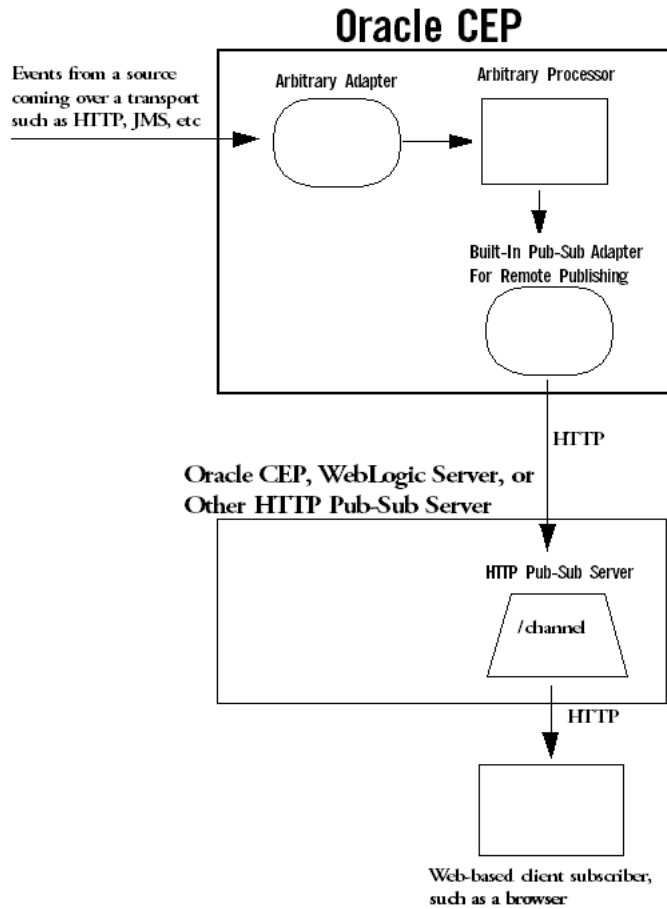
Note the following in Figure 7–2:

- Events flow from some source into an adapter of an application running in Oracle CEP. The arbitrary adapter is not required, it is shown only as an example.

- The events flow from the adapter to an arbitrary processor; again, this processor is not required.

- The processor sends the events to the built-in pub-sub adapter for remote publishing. The adapter in turn sends the events as messages to the remote HTTP pub-sub server for which the adapter is configured; this HTTP pub-sub server could be on another Oracle CEP instance, a WebLogic Server instance, or any other third-party implementation. The pub-sub adapter sends the messages to the channel for which it has been configured.

- The remote HTTP pub-sub server then sends the message to all subscribers of the channel.

7.1.2 Overview of the Built-In Pub-Sub Adapter for Subscribing

Figure 7–3 shows how the built-in pub-sub adapter for subscribing fits into a simple event processing network. The arbitrary processor and business POJO are not required, they are just an example of possible components in your application in addition to the pub-sub adapter.

Figure 7–3   Built-In Pub-Sub Adapter For Subscribing
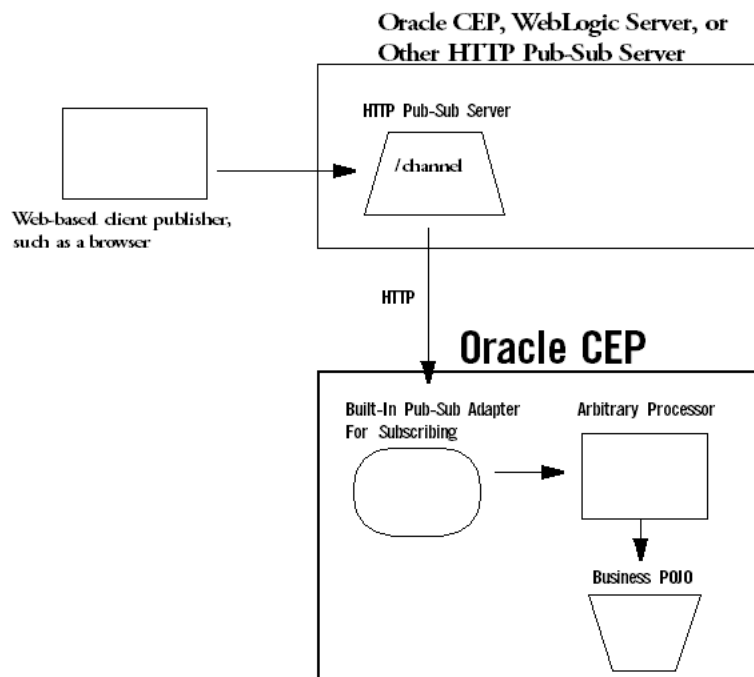
Note the following in Figure 7–3:

- Messages are published to a remote HTTP pub-sub server, which could be another instance of Oracle CEP, WebLogic Server, or a third-party implementation. The messages are typically published by Web based clients (shown in graphic), by the HTTP pub-sub server itself, or another server application.

- The built-in pub-sub adapter running in an Oracle CEP application subscribes to the HTTP pub-sub server and receives messages from the specified channel. The adapter converts the messages into the event type configured for the adapter.

- The pub-sub adapter sends the events to a processor. This processor is not required, it is shown only as an example of a typical Oracle CEP application.

- The processor sends the events to a business POJO. Again, this business POJO is not required.

For more information, see Section 7.1, "Overview of HTTP Publish-Subscribe Server Adapter Configuration".

7.1.3 Converting Between JSON Messages and Event Types

Oracle CEP can automatically convert incoming JSON messages to event types, and vice versa in the outbound case. However, if you want to customize the way a JSON message (either *inbound* via a HTTP pub-sub adapter for subscribing or *outbound* via an HTTP pub-sub adapter for publishing) is converted to an event type, or vice versa, you must create your own converter bean. See Section 7.2.1, "Creating a Custom Converter Between the HTTP Pub-Sub Messages and Event Types" for details.

If you do *not* provide your own converter class, and instead let Oracle CEP take care of the conversion between messages and event types, the following is true:

- You must specify an event type that Oracle CEP uses in its conversion. See Section 7.2.2, "Configuring an HTTP Pub-Sub Adapter" for details.

- The default converter used in the HTTP adapter for subscribing creates a new event of the specified type for each incoming message. For each property of the specified event type, it looks for a corresponding property name in the JSON object that constitutes the message, and if found, sets the corresponding value.

- The default converter used in the HTTP adapter for publishing creates a JSON message for each event. For each property of the specified event type, a corresponding element is created in the output JSON message.

7.2 Using the Built-In HTTP Pub-Sub Adapters in an Application

The following procedure describes typical steps for using the *built-in* pub-sub adapters in your Oracle CEP application. See Chapter 8, "Configuring Custom Adapters, Event Beans, and Spring Beans," for details about creating a custom pub-sub adapter.

> **Note:** This section assumes that you have already created an Oracle CEP application, along with its EPN assembly file and component configuration files, and that you want to update the application to use the built-in pub-sub adapters. If this is not true, refer to Chapter 1, "Overview of Creating Oracle CEP Applications" for general information about creating an Oracle CEP application.

To use the built-in HTTP pub-sub adapters in an application:

1. Optionally create a converter Java class if you want to customize the way the inbound or outbound messages are converted into event types. This step is optional because you can let Oracle CEP make the conversion based on mapping property names between the messages and a specified event type.

   See Section 7.2.1, "Creating a Custom Converter Between the HTTP Pub-Sub Messages and Event Types."

2. If you are going to use the local HTTP pub-sub server associated with the Oracle CEP instance for local publishing, use Visualizer, the Oracle CEP Administration Tool, to add new channels with the channel pattern required by your application.

   For details, see "How to Configure Security for an HTTP Publish-Subscribe Channel" in the *Oracle CEP Visualizer User's Guide*.

3. Configure the built-in pub-sub adapters you are going to add to you application by updating the component configuration files.

   See Section 7.2.2, "Configuring an HTTP Pub-Sub Adapter."

4. Update the EPN assembly file, adding declarations for each built-in pub-sub adapter you are adding to your application.

   See Section 7.2.3, "Updating the EPN Assembly File."

5. Update the `MANIFEST.MF` file of your application, adding the package `com.bea.core.encryption` to the `Import-Package` header. For example:

   ```
   Import-Package:
     com.bea.core.encryption
     com.bea.wlevs.adapter.defaultprovider;version="2.0.0.0",
     ...
   ```

   See Section 14.2.2.1, "Creating the MANIFEST.MF File" for additional information on the manifest file.

7.2.1 Creating a Custom Converter Between the HTTP Pub-Sub Messages and Event Types

If you want to customize the way a message (either *inbound* via a HTTP pub-sub adapter for subscribing or *outbound* via an HTTP pub-sub adapter for publishing) is converted to an event type, or vice versa, you must create your own converter bean.

The custom converter bean for an inbound HTTP pub-sub message must implement the `com.bea.wlevs.adapters.httppubsub.api.InboundMessageConverter` interface. This interface has a single method:

```
public List convert(JSONObject message) throws Exception;
```

The `message` parameter corresponds to the incoming HTTP pub-sub message and the return value is a `List` of events that will be passed on to the next stage of the event processing network. The incoming message is assumed to be the JSON format.

The custom converter bean for an outbound HTTP pub-sub message must implement the `com.bea.wlevs.adapters.httppubsub.api.OutboundMessageConverter` interface. This interface has a single method:

```
public List<JSONObject> convert(Object event) throws Exception;
```

The parameters correspond to an event received by the outbound HTTP pub-sub adapter from the source node in the EPN and the return value is a `List` of JSON messages.

See the *Oracle CEP Java API Reference* for a full description of these APIs.

The following example shows the Java source of a custom converter bean that implements both `InboundMessageConverter` and `OutboundMessageConvert`; this bean can be used for both inbound and outbound HTTP pub-sub adapters:

```
package com.sample.httppubsub;
import com.bea.wlevs.adapters.httppubsub.api.InboundMessageConverter;
import com.bea.wlevs.adapters.httppubsub.api.OutboundMessageConverter;
import com.bea.httppubsub.json.JSONObject;
import java.util.List;
import java.util.ArrayList;
import java.util.HashMap;
import java.util.Map;
public class TestConverter implements InboundMessageConverter, OutboundMessageConverter {
    public List convert(JSONObject message) throws Exception {
        List eventCollection = new ArrayList();
        PubsubTestEvent event = new PubsubTestEvent();
        event.setMessage("From TestConverter: " + message);
        eventCollection.add(event);
        return eventCollection;
    }
    public List<JSONObject> convert(Object event) throws Exception {
        List<JSONObject> list = new ArrayList<JSONObject>(1);
        Map map = new HashMap();
        map.put("message", ((PubsubTestEvent) event).getMessage());
        list.add(new JSONObject(map));
        return list;
    }
}
```

7.2.2 Configuring an HTTP Pub-Sub Adapter

You configure the built-in pub-sub adapters in their respective configuration files, similar to how you configure other components in the event processing network, such

as processors or streams. For general information about these configuration files, see Section 1.1.5, "Component Configuration Files."

The following configuration file shows a complete example of configuring each of the three built-in pub-sub adapters; the procedure following the example uses the example to show how to create your own file:

```xml
<?xml version="1.0" encoding="UTF-8"?>
<n1:config
  xsi:schemaLocation="http://www.bea.com/ns/wlevs/config/application wlevs_application_
config.xsd"
  xmlns:n1="http://www.bea.com/ns/wlevs/config/application"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
    <http-pub-sub-adapter>
        <name>remotePublisher</name>
        <server-url>http://myhost.com:9102/pubsub</server-url>
        <channel>/channel1</channel>
        <event-type>com.mycompany.httppubsub.PubsubEvent</event-type>
        <user>wlevs</user>
        <password>wlevs</password>
    </http-pub-sub-adapter>
    <http-pub-sub-adapter>
        <name>localPublisher</name>
        <server-context-path>/pubsub</server-context-path>
        <channel>/channel2</channel>
    </http-pub-sub-adapter>
    <http-pub-sub-adapter>
        <name>remoteSubscriber</name>
        <server-url>http://myhost.com:9102/pubsub</server-url>
        <channel>/channel3</channel>
        <event-type>com.mycompany.httppubsub.PubsubEvent</event-type>
    </http-pub-sub-adapter>
</n1:config>
```

The following procedure describes the main steps to configure the built-in pub-sub adapters for your application. For simplicity, it is assumed in the procedure that you are going to configure all components of an application in a single configuration XML file and that you have already created this file for your application.

See Section B.1, "Component Configuration Schema wlevs_application_config.xsd" for the complete XSD Schema that describes the configuration of the built-in pub-sub adapters.

To configure an HTTP pub-sub adapter:

1. Open the configuration XML file using your favorite XML editor.

2. For each built-in pub-sub adapter you want to configure, add a http-pub-sub-adapter child element of the config root element; use the <name> child element to uniquely identify it. This name value will be used later as the id attribute of the wlevs:adapter element in the EPN assembly file that defines the event processing network of your application. This is how Oracle CEP knows to which particular adapter in the EPN assembly file this adapter configuration applies.

   For example, assume your configuration file already contains a processor (contents removed for simplicity) and you want to configure instances of each of the three built-in pub-sub adapters; then the updated file might look like the following; details of the adapter configuration will be added in later steps:

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

```
xmlns:n1="http://www.bea.com/ns/wlevs/config/application"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
    <processor>
      ...
    </processor>
    <http-pub-sub-adapter>
        <name>remotePublisher</name>
        ...
    </http-pub-sub-adapter>
    <http-pub-sub-adapter>
        <name>remoteSubscriber</name>
        ...
    </http-pub-sub-adapter>
    <http-pub-sub-adapter>
        <name>localPublisher</name>
        ...
    </http-pub-sub-adapter>
</n1:config>
```

3. For each *remote* pub-sub adapter (for both publishing and subscribing), add a `server-url` child element of `http-pub-sub-adapter` to specify the URL of the *remote* HTTP pub-sub server to which the Oracle CEP application will publish or subscribe, respectively. The remote pub-sub server could be another instance of Oracle CEP, or a WebLogic Server instance, or it could be any third-party HTTP pub-sub server. For example:

```
<http-pub-sub-adapter>
    <name>remotePublisher</name>
        <server-url>http://myhost.com:9102/pubsub</server-url>
        ...
</http-pub-sub-adapter>
```

In the example, the URL of the remote HTTP pub-sub server to which the `remotePublisher` adapter will publish events is `http://myhost.com:9102/pubsub`.

4. For each *local* pub-sub adapter for publishing, add a `server-context-path` element to specify the path of the local HTTP pub-sub server associated with the Oracle CEP instance hosting the current Oracle CEP application.

By default, each Oracle CEP server is configured with an HTTP pub-sub server with path `/pubsub`; if, however, you have created a new local HTTP pub-sub server, or changed the default configuration, then specify the value of the `path` child element of the `http-pubsub` element in the server's `config.xml` file. For example:

```
<http-pub-sub-adapter>
    <name>localPublisher</name>
    <server-context-path>/pubsub</server-context-path>
    ...
</http-pub-sub-adapter>
```

5. For *all* the pub-sub adapters, whether they are local or remote or for publishing or subscribing, add a `channel` child element to specify the channel that the pub-sub adapter publishes or subscribes to, whichever is appropriate. For example:

```
<http-pub-sub-adapter>
    <name>localPublisher</name>
    <server-context-path>/pubsub</server-context-path>
    <channel>/channel2</channel>
</http-pub-sub-adapter>
```

In the example, the `localPublisher` pub-sub adapter publishes to a local channel with pattern `/channel2`.

6. For all pub-sub adapters for subscribing, add an `event-type` element that specifies the JavaBean to which incoming messages are mapped. You are required to specify this for all subscribing adapters. At runtime, Oracle CEP uses the incoming key-value pairs in the message to map the message data to the specified event type.

   You can also optionally use the `event-type` element in a pub-sub adapter for publishing if you want to limit the types of events that are published to just those specified by the `event-type` elements. Otherwise, all events sent to the pub-sub adapter are published. For example:

   ```
   <http-pub-sub-adapter>
       <name>remoteSubscriber</name>
       <server-url>http://myhost.com:9102/pubsub</server-url>
       <channel>/channel3</channel>
       <event-type>com.mycompany.httppubsub.PubsubEvent</event-type>
   </http-pub-sub-adapter>
   ```

   Be sure this event type has been registered in the EPN assembly file by specifying it as a child element of the `wlevs:event-type-repository` element.

7. Finally, if the HTTP pub-sub server to which the Oracle CEP application is publishing requires user authentication, add `user` and `password` (or `encrypted-password`) elements to specify the username and password or encrypted password. For example:

   ```
   <http-pub-sub-adapter>
       <name>remotePublisher</name>
       <server-url>http://myhost.com:9102/pubsub</server-url>
       <channel>/channel1</channel>
       <event-type>com.mycompany.httppubsub.PubsubEvent</event-type>
       <user>wlevs</user>
       <password>wlevs</password>
   </http-pub-sub-adapter>
   ```

7.2.3 Updating the EPN Assembly File

For each pub-sub adapter in your event processing network, you must add a corresponding `wlevs:adapter` tag to the EPN assembly file that describes the network; use the `provider` attribute to specify whether the adapter is for publishing or subscribing. Follow these guidelines:

- If you are using a built-in pub-sub adapter for publishing (either locally or remotely), set the `provider` attribute to `httppub`, as shown:

  ```
  <wlevs:adapter id="remotePublisher" provider="httppub"/>
  ```

  The value of the `id` attribute, in this case `remotePublisher`, must match the name specified for this built-in pub-sub adapter in its configuration file. Note that the declaration of the built-in adapter for publishing in the EPN assembly file does not specify whether this adapter is local or remote; you specify this in the adapter configuration file.

- If you are using a built-in pub-sub adapter for subscribing, set the provider attribute to `httpsub`, as shown:

  ```
  <wlevs:adapter id="remoteSubscriber" provider="httpsub"/>
  ```

The value of the `id` attribute, in this case `remoteSubscriber`, must match the name specified for this built-in pub-sub adapter in its configuration file.

As with any other stage in the EPN, add listeners to the `wlevs:adapter` element to integrate the pub-sub adapter into the event processing network. Typically, a pub-sub adapter for subscribing is the first stage in an EPN (because it receives messages) and a pub-sub adapter for publishing would be in a later stage (because it sends messages). However, the requirements of your own Oracle CEP application define where in the network the pub-sub adapters fit in.

Also be sure that the event types used by the pub-sub adapters have been registered in the event type repository using the `wlevs:event-type-repository` element.

The following sample EPN file shows an event processing network with two built-in pub-sub adapters for publishing both local and remote publishing); see the text after the example for an explanation:

```
<?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="com.mycompany.httppubsub.PubsubEvent">
           <wlevs:class>com.mycompany.httppubsub.PubsubEvent</wlevs:class>
       </wlevs:event-type>
   </wlevs:event-type-repository>
   <wlevs:adapter id="receiveFromFeed"
                  class="com.mycompany.httppubsub.ReceiveFromFeed">
   </wlevs:adapter>
   <wlevs:processor id="pubsubProcessor" />
   <wlevs:adapter id="remotePublisher" provider="httppub"/>
   <wlevs:adapter id="localPublisher" provider="httppub"/>
   <wlevs:channel id="feed2processor">
       <wlevs:source ref="receiveFromFeed"/>
       <wlevs:listener ref="pubsubProcessor"/>
   </wlevs:channel>
   <wlevs:channel id="pubsubStream">
       <wlevs:listener ref="remotePublisher"/>
       <wlevs:listener ref="localPublisher"/>
       <wlevs:source ref="pubsubProcessor"/>
   </wlevs:channel>
</beans>
```

In the preceding example:

- The `receiveFromFeed` adapter is a custom adapter that receives data from some data feed; the details of this adapter are not pertinent to this topic. The `receiveFromFeed` adapter then sends its events to the `pubsubProcessor` via the `feed2processor` channel.

- The `pubsubProcessor` processes the events from the `receiveFromFeed` adapter and then sends them to the `pubsubStream` channel, which in turn sends them to the two built-in pub-sub adapters: `remotePublisher` and `localPublisher`.

- Based on the configuration of these two pub-sub adapters (see examples in Section 7.2.2, "Configuring an HTTP Pub-Sub Adapter"), remotePublisher publishes events only of type com.mycompany.httppubsub.PubsubEvent and publishes them to the a channel called /channel1 on the HTTP pub-sub server hosted remotely at http://myhost.com:9102/pubsub.

  The localPublisher pub-sub adapter publishes all events it receives to the local HTTP pub-sub server, in other words, the one associated with the Oracle CEP server on which the application is running. The local pub-sub server's path is /pubsub and the channel to which the adapter publishes is called /channel2.

The following sample EPN file shows an event processing network with one built-in pub-sub adapter for subscribing; see the text after the example for an explanation:

```xml
<?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="com.mycompany.httppubsub.PubsubEvent">
            <wlevs:class>com.mycompany.httppubsub.PubsubEvent</wlevs:class>
        </wlevs:event-type>
    </wlevs:event-type-repository>
    <wlevs:adapter id="remoteSubscriber" provider="httpsub">
        <wlevs:listener ref="myEventBean"/>
    </wlevs:adapter>
    <bean id="myEventBean"
          class="com.mycompany.httppubsub.MyEventBean">
    </bean>
    <wlevs:channel id="pubsubStream" advertise="true">
        <wlevs:listener>
            <bean id="mySink"
                  class="com.mycompany.httppubsub.MySink"/>
        </wlevs:listener>
        <wlevs:source ref="myEventBean"/>
    </wlevs:channel>
</beans>
```

In the preceding example:

- The remoteSubscriber adapter is a built-in pub-sub adapter for subscribing.

  Based on the configuration of this adapter (see examples in Section 7.2.2, "Configuring an HTTP Pub-Sub Adapter"), remoteSubscriber subscribes to a channel called /channel3 configured for the remote HTTP pub-sub server hosted at http://myhost.com:9102/pubsub. Oracle CEP converts each messages it receives from this channel to an instance of com.mycompany.httppubsub.PubsubEvent and then sends it a Spring bean called myEventBean.

- The myEventBean processes the event as described by the com.mycompany.httppubsub.MyEventBean class, and then passes it the mySink event source via the pubsubStream channel. This section does not discuss the details of these components because they are not pertinent to the HTTP pub-sub adapter topic.

8

Configuring Custom Adapters, Event Beans, and Spring Beans

This section contains information on the following subjects:

8.1 Overview of Custom Adapters, Event Beans, and Spring Beans

This section provides an overview of adapter and event bean implementation, including:

8.1.1 Custom Adapters

One of the main roles of an adapter is to convert data coming from some channel, such as a market data feed, into Oracle CEP events. These events are then passed to other components in the application, such as processors. An adapter is usually the entry point to an Oracle CEP application. An adapter can also be the exit point of an

application so that it receives events from an intermediate component, converts the data into something that an external application can read, and then sends it out.

"Foreign Exchange (FX) Example" in the *Oracle CEP Getting Started* shows three adapters that read in data from currency data feeds and then pass the data, in the form of a specific event type, to the processors, which are the next components in the network.

You can create adapters of different types, depending on the format of incoming data and the technology you use in the adapter code to do the conversion. The most typical types of adapters are those that:

- Use a data vendor API, such as Reuters, Wombat, or Bloomberg.

- Convert incoming JMS messages using standard JMS APIs.

- Use other messaging systems, such as TIBCO Rendezvous.

- Use a socket connection to the customer's own data protocol.

Adapters are Java classes that implement specific Oracle CEP interfaces. You register the adapter classes in the EPN assembly file that describes your entire application.

You can optionally change the default configuration of the adapter, or even extend the configuration and add new configuration elements and attributes. There are two ways to pass configuration data to the adapter; the method you chose depends on whether you want to dynamically change the configuration after deployment:

If you are *not* going to change the configuration data after the adapter is deployed, then you can configure the adapter in the EPN assembly file.

If, however, you do want to be able to dynamically change the configuration elements, then you should put this configuration in the adapter-specific configuration files.

For more information, see Section 8.1, "Overview of Custom Adapters, Event Beans, and Spring Beans".

8.1.2 Custom Event Beans

Event beans are very similar to standard Spring beans except that they can be managed by the Oracle CEP management framework. Standard Spring beans are managed by the Spring framework. You register event beans in the EPN assembly file using the Oracle CEP `<wlevs:event-bean>` stage rather than the standard `<bean>` tag.

An event bean is a type of Stage, it can be monitored by the Oracle CEP monitoring framework, make use of the configuration metadata annotations, and it can be set to record, and play-back events that pass through it. An event bean can also participate in the Event Server bean lifecycle by specifying methods in its XML declaration, rather than by implementing Event Server API interfaces.

You can use either an event bean or a Spring bean:

- Use an event bean to actively use the capabilities of the Oracle CEP server container.

- Use a Spring bean for legacy integration to Spring.

For more information, see:

- Section 8.1.3, "Custom Spring Beans"

- Section 8.1.4, "Event Sources and Event Sinks"

- Section 8.1.5, "Adapter and Event Bean Factories"

8.1.3 Custom Spring Beans

Spring beans are managed by the Spring framework. You register Spring beans in the EPN assembly file using the standard `<bean>` tag.

A Spring bean cannot be monitored by the Oracle CEP monitoring framework, make cannot use the configuration metadata annotations, and cannot be set to recor, and play-back events that pass through it.

You can use either a Spring bean or an event bean:

- Use a Spring bean for legacy integration to Spring.
- Use an event bean to actively use the capabilities of the Oracle CEP server container.

For more information, see:

- Section 8.1.2, "Custom Event Beans"
- Section 8.1.4, "Event Sources and Event Sinks"
- Appendix A, "Additional Information about Spring and OSGi"

8.1.4 Event Sources and Event Sinks

Standard Spring beans and event beans can be event sources, event sinks, or both. Event sources generate events, event sinks receive events.

This section describes:

- Section 8.1.4.1, "Event Beans as Event Sources"
- Section 8.1.4.2, "Spring Beans as Event Sources"
- Section 8.1.4.3, "Event Beans as Event Sinks"
- Section 8.1.4.4, "Spring Beans as Event Sinks"

For more information, see:

- Section 8.3, "Implementing an Adapter or Event Bean as an Event Source"
- Section 8.4, "Implementing an Adapter or Event Bean as an Event Sink"

8.1.4.1 Event Beans as Event Sources

The implementation class of an event bean that is an event source may be specified as private to the application or as an advertised OSGI service re-usable by other applications.

For the framework to be able to fully manage the bean as an EPN component, it must be specified as an event-bean rather than a standard Spring bean. Management tasks include monitoring and record/playback.

You register event beans in the EPN assembly file using the <wlevs:event-bean> tag. For example:

```
<wlevs:event-bean id="recplayEventSink"
                  class="com.bea.wlevs.example.recplayRecplayEventSink">
    <wlevs:listener ref="playbackHttpPublisher"/>
</wlevs:event-bean>
```

8.1.4.2 Spring Beans as Event Sources

You specify that a standard Spring bean component in your EPN is an event source by implementing the `com.bea.wlevs.ede.api EventSource` API. The bean can also

optionally implement the various lifecycle interfaces, such as `InitializingBean`, `DisposableBean`, and the active interfaces, such as `RunnableBean`. If a Spring-bean implements `Runnable` but not `RunnableBean`, Oracle CEP does not run it in a thread. This is different behavior from an event bean.

The Spring bean event source can make use of the configuration metadata annotations, such as `@Prepare`, `@Rollback`, and `@Activate`.

You register the Spring bean in the EPN assembly file in the standard way using the `bean` element. You can then specify this bean as an event source of some other stage in the EPN.

8.1.4.3 Event Beans as Event Sinks

The functionality of event beans as event sinks is very similar to that of event sources except for the following:

- Event bean event sinks must implement the `EventSink` interface.

- Event bean event sinks must provide an `onevent-method`.

Event sinks are not active which means that if they implement the `Runnable` interface, Oracle CEP does not run them in a separate thread.

You reference event sinks in the EPN assembly file using the `wlevs:listener` element:

```
<wlevs:channel id="myStream" >
    <wlevs:listener ref="myEventSink" />
</wlevs:channel>
```

8.1.4.4 Spring Beans as Event Sinks

The functionality of Spring beans as event sinks is very similar to that of event sources except that Spring bean sinks must implement the `EventSink` interface.

Event sinks are not active which means that if they implement the `Runnable` interface, Oracle CEP does not run them in a separate thread.

You reference event sinks in the EPN assembly file using the `wlevs:listener` element:

```
<wlevs:channel id="myStream" >
    <wlevs:listener ref="myEventSink" />
</wlevs:channel>
```

8.1.5 Adapter and Event Bean Factories

If your adapter or event bean is going to be used only by a single Oracle CEP application, then you do not need to create a factory. However, if multiple applications are going to use the same adapter or event bean, then you should also program a factory. In this case, every application gets its own instance of the adapter.

Adapter or event bean factories must implement the `com.bea.wlevs.ede.api.Factory` interface. This interface has a single method, `create()`, that you implement to create an adapter or event bean instance.

You register factories in the EPN assembly file using the `wlevs:factory` element:

```
<wlevs:factory provider-name="myprovider" class="my.Implementation"/>
```

Note that if you need to specify service properties, then you must use the `<osgi:service>` tag to register the factory.

For more information, see Section 8.5, "Implementing an Adapter or Event Bean Factory"

8.2 Implementing an Adapter or Event Bean

The following procedure describes the typical steps for creating an adapter.

To implement an adapter or event bean:

1. Program the adapter or event bean Java class.

    If your adapter or event bean is an event source, see Section 8.3, "Implementing an Adapter or Event Bean as an Event Source."

    If your adapter or event bean is an event sink, see Section 8.4, "Implementing an Adapter or Event Bean as an Event Sink."

2. Optionally program the factory class. You only need to do this if many applications are going to use the adapter or event bean.

    See Section 8.5, "Implementing an Adapter or Event Bean Factory."

3. Update the EPN assembly file with adapter, event bean, and adapter factory registration info.

    See Section 8.7, "Updating the EPN Assembly File."

4. Optionally change the default configuration of the adapter.

    See Section 8.8, "Configuring an Adapter or Event Bean."

5. Optionally extend the configuration of the adapter if its basic one is not adequate.

    See Section 8.9, "Extending the Configuration of an Adapter or Event Bean."

In the preceding procedure, it is assumed that the adapter or event bean is bundled in the same application JAR file that contains the other components of the event network, such as the processor, streams, and business logic POJO. If you want to bundle the adapter or event bean in its own JAR file so that it can be shared among many applications, see Section 8.11, "Assembling an Adapter or Event Bean in Its Own Bundle."

8.3 Implementing an Adapter or Event Bean as an Event Source

This section describes how to create an inbound adapter that acts as an event source because it receives incoming data and generates events that it sends to the next component in the EPN. Because event beans are functionally the same, the guidelines also apply to programming event beans that act as event sources.

The inbound adapter class typically reads the stream of incoming data, such as from a market data feed, converts it into an Oracle CEP event type that is understood by the rest of the application, and sends the event to the next component in the network.

The following example shows the adapter class of the HelloWorld sample; see the explanation after the example for coding guidelines that correspond to the Java code in bold.

```
package com.bea.wlevs.adapter.example.helloworld;
import java.util.ArrayList;
import java.util.Date;
import java.util.List;
import com.bea.wlevs.ede.api.EventSender;
import com.bea.wlevs.ede.api.EventSource;
```

```java
import com.bea.wlevs.ede.api.RunnableBean;
import com.bea.wlevs.event.example.helloworld.HelloWorldEvent;
public class HelloWorldAdapter implements RunnableBean, EventSource {
    private static final int SLEEP_MILLIS = 300;
    private DateFormat dateFormat;
    private String message;
    private EventSender eventSender;
    private boolean stopped;
    public HelloWorldAdapter() {
        super();
        dateFormat = DateFormat.getTimeInstance();
    }
    public void run() {
        stopped = false;
        while (!isStopped()) { // Generate messages forever...
            generateHelloMessage();
            try {
                synchronized (this) {
                    wait(SLEEP_MILLIS);
                }
            } catch (InterruptedException e) {
                e.printStackTrace();
            }
        }
    }
    public void setMessage(String message) {
        this.message = message;
    }
    @SuppressWarnings({ "unchecked" })
    private void generateHelloMessage() {
        List eventCollection = new ArrayList();
        String message = this.message + dateFormat.format(new Date());
        HelloWorldEvent event = new HelloWorldEvent();
        event.setMessage(message);
        eventCollection.add(event);
        eventSender.sendEvent(eventCollection, null);
    }
    public void setEventSender(EventSender sender) {
        eventSender = sender;
    }
    public synchronized void suspend() {
        stopped = true;
    }
    private synchronized boolean isStopped() {
        return stopped;
    }
}
```

Follow these guidelines when programming the adapter Java class; code snippets of
the guidelines are shown in bold in the preceding example:

- Import the interfaces and classes of the Oracle CEP API:

```java
import com.bea.wlevs.ede.api.EventSender;
import com.bea.wlevs.ede.api.EventSource;
import com.bea.wlevs.ede.api.RunnableBean;
```

  Because the adapter is an event source it must implement the EventSource
  interface. If you want the adapter to run in a thread, also implement
  RunnableBean. The EventSender interface sends event types to the next
  component in your application network. For full details of these APIs, see *Oracle
  CEP Java API Reference*.

- Import the application-specific classes that represent the event types used in the
  application:

```
import com.bea.wlevs.event.example.helloworld.HelloWorldEvent
```

The `com.bea.wlevs.event.example.helloworld.HelloWorldEvent` class is a JavaBean that represents the event type used in the application.

- The adapter class must implement the EventSource and `RunnableBean` interfaces because it is an event source and will run in its own thread:

```
public class HelloWorldAdapter implements RunnableBean, EventSource {
```

The `EventSource` interface provides the `EventSender` that you use to send events.

- Because the adapter implements the `RunnableBean` interface, your adapter must then implement the `run()` method:

`public void run() {``...`

This is where you should put the code that reads the incoming data, such as from a market feed, and convert it into an Oracle CEP event type, and then send the event to the next component in the network. Refer to the documentation of your data feed provider for details on how to read the incoming data. See Section 14.2.2.2, "Accessing Third-Party JAR Files" for information about ensuring you can access the vendor APIs if they are packaged in a third-party JAR file.

In the HelloWorld example, the adapter itself generates the incoming data using the `generateHelloMessage()` private method. This is just for illustrative purposes and is not a real-world scenario. The `generateHelloMessage()` method also includes the other typical event type programming tasks:

```
HelloWorldEvent event = new HelloWorldEvent();
event.setMessage(message);
eventCollection.add(event);
eventSender.sendEvent(eventCollection, null);
```

The `HelloWorldEvent` is the event type used by the HelloWorld example; the event type is implemented with a JavaBean and is registered in the EPN assembly file using the `<wlevs:event-type-repository>` tag. See Section 1.5, "Creating Oracle CEP Event Types" for details. The `setMessage()` method sets the properties of the event; in typical adapter implementations this is how you convert a particular property of the incoming data into an event type property. Finally, the `EventSender.sendEvent()` method sends this new event to the next component in the network.

- Because your adapter implements `EventSource`, you must implement the `setEventSender()` method, which passes in the `EventSender` that you use to send events:

```
public void setEventSender(EventSender sender) { ...
```

- If, as is typically the case, your adapter implements `SuspendableBean`, you must implement the `suspend()` method that stops the adapter when, for example, the application is undeployed:

```
public synchronized void suspend() throws Exception { ...
```

8.4 Implementing an Adapter or Event Bean as an Event Sink

The following sample code shows a Spring bean from HelloWorld application that acts as an event sink; see the explanation after the example for the code shown in bold:

```
package com.bea.wlevs.example.helloworld;
import java.util.List;
import com.bea.wlevs.ede.api.EventRejectedException;
import com.bea.wlevs.ede.api.EventSink;
import com.bea.wlevs.event.example.helloworld.HelloWorldEvent;
public class HelloWorldBean implements EventSink {
    @SuppressWarnings({ "unchecked" })
    public void onEvent(List newEvents)
            throws EventRejectedException {
        for (Object event : newEvents) {
            if (event instanceof HelloWorldEvent) {
                HelloWorldEvent helloWorldEvent = (HelloWorldEvent) event;
                System.out.println("Message: " + helloWorldEvent.getMessage());
            }
        }
    }
}
```

The programming guidelines shown in the preceding example are as follows:

- Your bean must import the event type of the application, which in the HelloWorld case is HelloWorldEvent:

  ```
  import com.bea.wlevs.event.example.helloworld.HelloWorldEvent;
  ```

- Your bean must implement the com.bea.wlevs.ede.api.EventSink interface:

  ```
  public class HelloWorldBean implements EventSink {...
  ```

- The EventSink interface has a single method that you must implement, onEvent(java.util.List), which is a callback method for receiving events. The parameter of the method is a List that contains the actual events that the bean received from the component that sent it the events:

  ```
  public void onEvent(List newEvents)
  ```

- The data type of the events is determined by the event type you registered in the EPN assembly file of the application. In the example, the event type is HelloWorldEvent; the code first ensures that the received event is truly a HelloWorldEvent:

  ```
  if (event instanceof HelloWorldEvent) {
      HelloWorldEvent helloWorldEvent = (HelloWorldEvent) event;
  ```

  This event type is a JavaBean that was configured in the EPN assembly file as shown:

  ```
  <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>
  ```

  See Section 1.4, "Creating the EPN Assembly File" for procedural information about creating the EPN assembly file, and Appendix D, "Schema Reference: EPN Assembly spring-wlevs-v11_0_0_0.xsd" for reference information.

- Events are instances of the appropriate JavaBean, so you access the individual properties using the standard getXXX() methods. In the example, the HelloWorldEvent has a property called message:

```
                System.out.println("Message: " + helloWorldEvent.getMessage());
```

For complete API reference information about the Oracle CEP APIs described in this section, see the *Oracle CEP Java API Reference*.

8.5 Implementing an Adapter or Event Bean Factory

Your adapter factory class must implement the `com.bea.wlevs.ede.api.AdapterFactory` interface, which has a single method, `create()`, in which you code the creation of your specific adapter class. Event beans implement `Factory`.

The following is a possible adapter factory class for the HelloWorld example:

```
package com.bea.adapter.wlevs.example.helloworld;
import com.bea.wlevs.ede.api.Adapter;
import com.bea.wlevs.ede.api.AdapterFactory;
public class HelloWorldAdapterFactory implements Factory {
    public HelloWorldAdapterFactory() {
    }
    public synchronized Adapter create() throws IllegalArgumentException {
        return new HelloWorldAdapter();
    }
}
```

For full details of these APIs, see the *Oracle CEP Java API Reference*.

8.6 Accessing a Relational Database

You can use the Java Database Connectivity (JDBC) APIs (`http://java.sun.com/javase/technologies/database/`) in your adapters, event beans, and standard Spring beans to access data contained in a relational database. Oracle CEP supports JDBC 3.0. For more information, see `http://java.sun.com/products/jdbc/download.html#corespec30`.

To access a relational database:

1. Configure JDBC for Oracle CEP.

    For more information, see "Configuring JDBC for Oracle CEP" in the *Oracle CEP Administrator's Guide*.

2. In your bean Java code, you can start using the JDBC APIs as usual, by using a `DataSource` or instantiating a `DriverManager`. For example:

    ```
    OracleDataSource ods = new OracleDataSource();
    ods.setURL("jdbc:oracle:thin:user/passwd@localhost:1521/XE");
    Connection conn = ods.getConnection();
    ```

    For additional programming information, see `http://java.sun.com/j2se/1.5.0/docs/guide/jdbc/getstart/GettingStartedTOC.fm.html`.

8.7 Updating the EPN Assembly File

The adapters, event beans, and adapter factory must be registered in the EPN assembly file, as discussed in the following sections:

■ Section 8.7.1, "Registering the Adapter or Event Bean Factory"

■ Section 8.7.2, "Declaring the Adapter and Event Bean Components in your Application"

For a complete description of the configuration file, including registration of other components of your application, see Section 1.4, "Creating the EPN Assembly File."

8.7.1 Registering the Adapter or Event Bean Factory

You register factories in the EPN assembly file using the `wlevs:factory` element:

```
<wlevs:factory provider-name="myprovider" class="my.Implementation"/>
```

If you need to specify service properties, then you must use the `osgi:service` element to register the factory as an OSGI service in the EPN assembly file. The scope of the OSGI service registry is the entire Oracle CEP. This means that if more than one application deployed to a given server is going to use the same adapter factory, be sure to register the adapter factory only *once* as an OSGI service.

Add an entry to register the service as an implementation of the `com.bea.wlevs.ede.api.AdapterFactory` interface. Provide a property, with the `key` attribute equal to `type`, and the name by which this adapter provider will be referenced. Finally, add a nested standard Spring `bean` element to register the your specific adapter class in the Spring application context

For example, the following segment of the EPN assembly file registers the `HelloWorldAdapterFactory` as the provider for type `hellomsgs`:

```
<osgi:service interface="com.bea.wlevs.ede.api.AdapterFactory">
    <osgi:service-properties>
        <entry key="type" value="hellomsgs"></entry>
    </osgi:service-properties>
    <bean  class="com.bea.adapter.wlevs.example.helloworld.HelloWorldAdapterFactory" />
</osgi:service>
```

8.7.2 Declaring the Adapter and Event Bean Components in your Application

In the EPN assembly file, you use the `wlevs:adapter` element to declare an adapter as a component in the event processor network. Similarly, you use the `wlevs:event-bean` element for event beans. For example:

```
<wlevs:event-bean id="recplayEventSink"
                  class="com.bea.wlevs.example.recplayRecplayEventSink">
    <wlevs:listener ref="playbackHttpPublisher"/>
</wlevs:event-bean>
```

If you registered an optional factory as an OSGI service, then use the `provider` attribute to point to the name you specified as the `type` in your `osgi:service` entry; for example:

```
<wlevs:adapter id="helloworldAdapter" provider="hellomsgs"/>
```

This means that an adapter will be instantiated by the factory registered for the type `hellomsgs`.

You can also use a `wlevs:instance-property` child element of `wlevs:adapter` to set any *static* properties in the adapter bean. Static properties are those that you will not dynamically change after the adapter is deployed.

For example, if your adapter class has a `setPort()` method, you can pass it the port number as shown:

```
<wlevs:adapter id="myAdapter" provider="myProvider">
    <wlevs:instance-property name="port" value="9001" />
```

```
</wlevs:adapter>
```

8.8 Configuring an Adapter or Event Bean

This section applies to both adapters and event beans. For simplicity the text mentions only adapters. The configuration of an event bean would be enclosed in the `event-bean` element.

Each adapter in your application has a default configuration.

The default adapter configuration is typically adequate for most applications. However, if you want to change this configuration, you must create an XML file that is deployed as part of the Oracle CEP application bundle. You can later update this configuration at runtime using the `wlevs.Admin` utility or manipulating the appropriate JMX Mbeans directly.

For more information, see:

- "wlevs.Admin Command-Line Reference" in the *Oracle CEP Administrator's Guide*

- "Configuring JMX for Oracle CEP" in the *Oracle CEP Administrator's Guide*

If your application has more than one adapter, you can create separate XML files for each adapter, or create a single XML file that contains the configuration for all adapters, or even all components of your application (adapters, processors, and streams). Choose the method that best suits your development environment.

The following procedure describes the main steps to create the adapter configuration file. For simplicity, it is assumed in the procedure that you are going to configure all components of an application in a single XML file

See Section B.1, "Component Configuration Schema wlevs_application_config.xsd" for the complete XSD Schema that describes the adapter configuration file.

To configure an adapter or event bean:

1. Create an XML file using your favorite XML editor. You can name this XML file anything you want, provided it ends with the `.xml` extension.

   The root element of the configuration file is `config`, with namespace definitions shown in the next step.

2. For each adapter in your application, add an `adapter` child element of `config`.

   Uniquely identify each adapter with the `name` child element. This name must be the same as the value of the `id` attribute in the `wlevs:adapter` element of the EPN assembly file that defines the event processing network of your application. This is how Oracle CEP knows to which particular adapter component in the EPN assembly file this adapter configuration applies. See Section 1.4, "Creating the EPN Assembly File" for details.

   For example, if your application has two adapters, the configuration file might initially look like this:

```
<?xml version="1.0" encoding="UTF-8"?>
<helloworld:config
  xmlns:helloworld="http://www.bea.com/xml/ns/wlevs/example/helloworld">
  <processor>
   ...
  </processor>
  <adapter>
    <name>firstAdapter</name>
    ...
```

```
    </adapter>
    <adapter>
      <name>secondAdapter</name>
      ...
    </adapter>
</helloworld:config>
```

In the example, the configuration file includes two adapters called `firstAdapter` and `secondAdapter`. This means that the EPN assembly file must include at least two adapter registrations with the same identifiers:

```
<wlevs:adapter id="firstAdapter" ...>
  ...
</wlevs:adapter>
<wlevs:adapter id="secondAdapter" ...>
  ...
</wlevs:adapter>
```

> **Caution:**   Identifiers and names in XML files are case sensitive, so be sure you specify the same case when referencing the component's identifier in the EPN assembly file.

8.8.1 Example of an Adapter Configuration File

The following sample XML file shows how to configure two adapters, `firstAdapter` and `secondAdapter`.

```
<?xml version="1.0" encoding="UTF-8"?>
<sample:config
  xmlns:sample="http://www.bea.com/xml/ns/wlevs/example/sample">
  <adapter>
    <name>firstAdapter</name>
  </adapter>
  <adapter>
    <name>secondAdapter</name>
  </adapter>
</sample:config>
```

8.9 Extending the Configuration of an Adapter or Event Bean

This section applies to both adapters and event beans. For simplicity the text mentions only adapters.

Adapters have default configuration data, as described in Section 8.8, "Configuring an Adapter or Event Bean" and Section B.1, "Component Configuration Schema wlevs_application_config.xsd". This default configuration is typically adequate for simple and basic applications.

However, you can also extend this configuration by using XSD Schema to specifying a *new* XML format of an adapter configuration file that extends the built-in XML type provided by Oracle CEP. By extending the XSD Schema, you can add as many new elements to the adapter configuration as you want, with few restrictions other than each new element must have a `name` attribute. This feature is based on standard technologies, such as XSD Schema and Java Architecture for XML Binding (JAXB). For more information, see `https://jaxb.dev.java.net/`.

To extend the configuration of an adapter or event bean:

1. Create the new XSD Schema file that describes the extended adapter configuration. This XSD file must also include the description of the other components in your application (processors and streams), although you typically use built-in XSD types, defined by Oracle CEP, to describe them.

   See Section 8.9.1, "Creating the XSD Schema File" for details.

2. As part of your application build process, generate the Java representation of the XSD schema types using a JAXB binding compiler, such as the `com.sun.tools.xjc.XJCTask` Ant task (see `https://jaxb.dev.java.net/jaxb20-ea/docs/xjcTask.html`) from Sun's GlassFish reference implementation. This Ant task is included in the Oracle CEP distribution for your convenience.

   The following sample `build.xml` file shows how to do this:

```
<property name="base.dir" value="." />
<property name="output.dir" value="output" />
<property name="sharedlib.dir" value="${base.dir}/../../../../../modules" />
<property name="wlrtlib.dir" value="${base.dir}/../../../../modules"/>
<path id="classpath">
        <pathelement location="${output.dir}" />
        <fileset dir="${sharedlib.dir}">
                <include name="*.jar" />
        </fileset>
        <fileset dir="${wlrtlib.dir}">
                <include name="*.jar"/>
        </fileset>
</path>
<taskdef name="xjc" classname="com.sun.tools.xjc.XJCTask">
    <classpath refid="classpath" />
</taskdef>
<target name="generate" depends="clean, init">
   <copy file="../../../../xsd/wlevs_base_config.xsd"
        todir="src/main/resources/extension" />
   <copy file="../../../../xsd/wlevs_application_config.xsd"
         todir="src/main/resources/extension" />
   <xjc extension="true" destdir="${generated.dir}">
     <schema dir="src/main/resources/extension"
             includes="helloworld.xsd"/>
     <produces dir="${generated.dir}" includes="**/*.java" />
   </xjc>
</target>
```

   In the example, the extended XSD file is called `helloworld.xsd`. The build process copies the Oracle CEP XSD files (`wlevs_base_config.xsd` and `wlevs_application_config.xsd`) to the same directory as the `helloworld.xsd` file because `helloworld.xsd` imports the Oracle CEP XSD files.

3. Compile these generated Java files into classes.

4. Package the compiled Java class files in your application bundle.

   See Section 14.2, "Assembling an Oracle CEP Application" for details.

5. Program your adapter as described in Section 8.3, "Implementing an Adapter or Event Bean as an Event Source." Within your adapter code, you access the extended configuration as usual, as described in Section 8.9.3, "Programming Access to the Configuration of an Adapter or Event Bean."

6. When you create the configuration XML file that describes the components of your application, be sure you use the extended XSD file as its description. In addition, be sure you identify the namespace for this schema rather than the default schema. For example:

```xml
<?xml version="1.0" encoding="UTF-8"?>
<helloworld:config
  xmlns:helloworld="http://www.bea.com/xml/ns/wlevs/example/helloworld">
  <adapter>
    <name>helloworldAdapter</name>
    <message>HelloWorld - the current time is:</message>
  </adapter>
</helloworld:config>
```

8.9.1 Creating the XSD Schema File

The new XSD schema file extends the `wlevs_application_config.xsd` XSD schema (see Section B.1, "Component Configuration Schema wlevs_application_config.xsd") and then adds new custom information, such as new configuration elements for an adapter. Use standard XSD schema syntax for your custom information.

Oracle recommends that you use the XSD schema in Section 8.9.2, "Complete Example of an Extended XSD Schema File" as a basic template, and modify the content to suit your needs. In addition to adding new configuration elements, other modifications include changing the package name of the generated Java code and the element name for the custom adapter. You can control whether the schema allows just your custom adapter or other components like processors.

To create a new XSD schema file:

1. Using your favorite XML Editor, create the basic XSD file with the required namespaces, in particular those for JAXB. For example:

```xml
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema targetNamespace="http://www.bea.com/xml/ns/wlevs/example/helloworld"
        xmlns="http://www.bea.com/xml/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/xml/ns/wlevs/config/application"
        jxb:extensionBindingPrefixes="xjc" jxb:version="1.0"
        elementFormDefault="unqualified" attributeFormDefault="unqualified">
...
</xs:schema>
```

2. Import the `wlevs_application_config.xsd` XSD schema:

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

The `wlevs_application_config.xsd` in turn imports the `wlevs_base_config.xsd` XSD file.

3. Use the `complexType` XSD element to describe the XML type of the extended adapter configuration.

The new type must extend the `AdapterConfig` type, defined in `wlevs_application_config.xsd`. `AdapterConfig` extends `ConfigurationObject`. You can then add new elements or attributes to the

basic adapter configuration as needed. For example, the following type called `HelloWorldAdapterConfig` adds a `message` element to the basic adapter configuration:

```
<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>
```

4. Define a top-level element that *must* be named `config`.

In the definition of the `config` element, define a sequence of child elements that correspond to the components in your application. Typically the name of the elements should indicate what component they configure (`adapter`, `processor`, `channel`) although you can name then anything you want.

Each element must extend the `ConfigurationObject` XML type, either explicitly using the `xs:extension` element with `base` attribute value `base:ConfigurationObject` or by specifying an XML type that itself extends `ConfigurationObject`. The `ConfigurationObject` XML type, defined in `wlevs_base_config.xsd`, defines a single attribute: `name`.

The type of your adapter element should be the custom one you created in a preceding step of this procedure.

You can use the following built-in XML types that `wlevs_application_config.xsd` describes, for the child elements of `config` that correspond to processors or streams:

- `DefaultProcessorConfig`—See Section 11.1, "Overview of EPL Processor Component Configuration" for a description of the default processor configuration.

- `DefaultStreamConfig`—See Section 9.1, "Overview of Channel Configuration" for a description of the default channel configuration.

For example:

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

5. Optionally use the `jxb:package` child element of `jxb:schemaBindings` to specify the package name of the generated Java code:

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

8.9.2 Complete Example of an Extended XSD Schema File

Use the following extended XSD file as a template:

```xml
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema targetNamespace="http://www.bea.com/xml/ns/wlevs/example/helloworld"
        xmlns="http://www.bea.com/xml/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/xml/ns/wlevs/config/application"
        jxb:extensionBindingPrefixes="xjc" jxb:version="1.0"
        elementFormDefault="unqualified" attributeFormDefault="unqualified">
        <xs:annotation>
                <xs:appinfo>
                        <jxb:schemaBindings>
                                <jxb:package
name="com.bea.adapter.wlevs.example.helloworld"/>
                        </jxb:schemaBindings>
                </xs:appinfo>
        </xs:annotation>
        <xs:import namespace="http://www.bea.com/xml/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="channel" 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>
```

8.9.3 Programming Access to the Configuration of an Adapter or Event Bean

This section applies to both adapters and event beans. For simplicity the text mentions only adapters.

When you deploy your application, Oracle CEP maps the configuration of each component (specified in the component configuration XML files) into Java objects using the Java Architecture for XML Binding (JAXB) standard (for more information, see https://jaxb.dev.java.net/). Because there is a single XML element that contains the configuration data for each component, JAXB in turn also produces a single Java class that represents this configuration data. Oracle CEP passes an instance of this Java class to the component (processor, channel, or adapter) at runtime when the component is initialized, and also whenever there is a dynamic change to the component's configuration.

In your adapter implementation, you can use metadata annotations to specify the Java methods that are invoked by Oracle CEP at runtime. Oracle CEP passes an instance of the configuration Java class to the specified methods; you can then program these methods to get specific runtime configuration information about the adapter. The following example shows how to annotate the activateAdapter() method with

the `@Activate` annotation to specify the method invoked when the adapter configuration is first activated:

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

By default, the date type of the adapter configuration Java class is `com.bea.wlevs.configuration.application.DefaultAdapterConfig`. If, however, you have extended the configuration of your adapter by creating your own XSD file that describes the configuration XMLfile, then you specify the type in the XSD file. In the preceding example, the data type of the Java configuration object is `com.bea.wlevs.example.helloworld.HelloWorldAdapterConfig`.

The metadata annotations provided are as follows:

- `com.bea.wlevs.management.Activate`—Specifies the method invoked when the configuration is activated.

  See Section G.2, "com.bea.wlevs.management.Activate" for additional details about using this annotation in your adapter code.

- `com.bea.wlevs.management.Prepare`—Specifies the method invoked when the configuration is prepared.

  See Section G.3, "com.bea.wlevs.management.Prepare" for additional details about using this annotation in your adapter code.

- `com.bea.wlevs.management.Rollback`—Specifies the method invoked when the adapter is terminated due to an exception.

  See Section G.4, "com.bea.wlevs.management.Rollback" for additional details about using this annotation in your adapter code.

8.10 Passing Login Credentials from an Adapter to the Data Feed Provider

If your adapter accesses an external data feed, the adapter might need to pass login credentials (username and password) to the data feed for user authentication.

The simplest, and least secure, way to do this is to hard-code the non-encrypted login credentials in your adapter Java code. However, this method does not allow you to encrypt the password or later change the login credentials without recompiling the adapter Java code.

The following procedures describe a different method that takes these two issues into account. In the procedure, it is assumed that the username to access the data feed is `juliet` and the password is `superSecret`.

You must decide whether you want the login credentials to be configured statically in the EPN assembly file, or dynamically by extending the configuration of the adapter. Configuring the credentials statically in the EPN assembly file is easier, but if the credentials later change you must restart the application for the update to the EPN assembly file to take place. Extending the adapter configuration allows you to change the credentials dynamically without restarting the application, but extending the configuration involves additional steps, such as creating an XSD file and compiling it into a JAXB object.

This section describes:

- Section 8.10.1, "How to Pass Static Login Credentials from an Adapter to the Data Feed Provider"

- Section 8.10.2, "How to Pass Dynamic Login Credentials from an Adapter to the Data Feed Provider"

- Section 8.10.3, "Updating the Adapter Code to Access the Login Credential Properties"

8.10.1 How to Pass Static Login Credentials from an Adapter to the Data Feed Provider

This procedure describes how to pass login credentials that you configure statically in the EPN assembly file.

To pass static credentials from an adapter to the data feed provider:

1. Open a command window and set your environment as described in "Setting Your Development Environment" in the *Oracle CEP Getting Started*.

2. Change to the directory that contains the EPN assembly file for your application.

3. Using your favorite XML editor, edit the EPN assembly file by updating the `wlevs:adapter` element that declares your adapter.

   In particular, add two instance properties that correspond to the username and password of the login credentials. For now, specify the cleartext password value; you will encrypt it in a later step. Also add a temporary `password` element whose value is the cleartext password. For example:

   ```
   <wlevs:adapter id="myAdapter" provider="myProvider">
     <wlevs:instance-property name="user" value="juliet"/>
     <wlevs:instance-property name="password" value="superSecret"/>
     <password>superSecret</password>
   </wlevs:adapter>
   ```

4. Save the EPN assembly file.

5. Use the `encryptMSAConfig` command to encrypt the value of the `password` element in the EPN assembly file:

   ```
   prompt> ORACLE_CEP_HOME/ocep_11.1/bin/encryptMSAConfig . epn_assembly_file
   msainternal.dat_file
   ```

   where *ORACLE_CEP_HOME* refers to the main directory into which you installed Oracle CEP, such as `d:\oracle_cep`. The second argument refers to the directory that contains the EPN assembly file; because this procedure directs you to change to the directory, the example shows " . ". The *epn_assembly_file* parameter refers to the name of your EPN assembly file. Finally, the *msainternal.dat_file* parameter refers to the location of the `.msainternal.dat` file associated with your domain; by default this file is located in the *DOMAIN_DIR*/*servername* directory, where *DOMAIN_DIR* refers to the domain directory such as `/oracle_cep/user_projects/domains/mydomain` and *servername* refers to the server instance.

   For more information, see "The encryptMSAConfig Command-Line Utility" in the *Oracle CEP Administrator's Guide*.

   After you run the command, the value of the `password` element of the EPN assembly file will be encrypted.

6. Edit the EPN assembly file:

Copy the encrypted value of the `password` element to the `value` attribute of the `password` instance property.

Remove the `password` element from the XML file.

For example:

```
<wlevs:adapter id="myAdapter" provider="myProvider">
    <wlevs:instance-property name="user" value="juliet"/>
    <wlevs:instance-property name="password"
        value="{Salted-3DES}B7L6nehu7dgPtJJTnTJWRA=="/>
</wlevs:adapter>
```

7. Update your adapter Java code to access the login credentials properties you have just configured and decrypt the password.

   See Section 8.10.1, "How to Pass Static Login Credentials from an Adapter to the Data Feed Provider."

8. Edit the `MANIFEST.MF` file of the application and add the `com.bea.core.encryption` package to the `Import-Package` header. See Section 14.2.2.1, "Creating the MANIFEST.MF File."

9. Re-assemble and deploy your application as usual. See Chapter 14, "Assembling and Deploying Oracle CEP Applications."

8.10.2 How to Pass Dynamic Login Credentials from an Adapter to the Data Feed Provider

This procedure describes how to pass login credentials that you configure dynamically by extending the configuration of the adapter.

To pass dynamic login credentials from an adapter to the data feed provider:

1. Extend the configuration of your adapter by adding two new elements: `user` and `password`, both of type `string`.

   For example, if you were extending the adapter in the HelloWorld example, the XSD file might look like the following:

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

   See Section 8.9, "Extending the Configuration of an Adapter or Event Bean" for detailed instructions.

2. Open a command window and set your environment as described in "Setting Your Development Environment" in the *Oracle CEP Getting Started*.

3. Change to the directory that contains the component configuration XML file for your adapter.

4. Using your favorite XML editor, update this component configuration XML file by adding the required login credentials using the `<user>` and `<password>`

elements. For now, specify the cleartext password value; you will encrypt it in a later step. For example:

```
<?xml version="1.0" encoding="UTF-8"?>
<myExample:config
  xmlns:myExample="http://www.bea.com/xml/ns/wlevs/example/myExample">
  <adapter>
    <name>myAdapter</name>
    <user>juliet</user>
    <password>superSecret</password>
  </adapter>
</myExample:config>
```

5. Save the adapter configuration file.

6. Use the `encryptMSAConfig` command to encrypt the value the `password` element in the adapter configuration file:

   ```
   prompt> ORACLE_CEP_HOME/ocep_11.1/bin/encryptMSAConfig . adapter_config_file
   msainternal.dat_file
   ```

   where `ORACLE_CEP_HOME` refers to the main directory into which you installed Oracle CEP, such as `d:\oracle_cep`. The second argument refers to the directory that contains the adapter configuration file; because this procedure directs you to change to the directory, the example shows `"."`. The `adapter_config_file` parameter refers to the name of your adapter configuration file. Finally, the `msainternal.dat_file` parameter refers to the location of the `.msainternal.dat` file associated with your domain; by default this file is located in the `DOMAIN_DIR/servername` directory, where `DOMAIN_DIR` refers to the domain directory such as `/oracle_cep/user_projects/domains/mydomain` and `servername` refers to the server instance.

   For more information, see "The encryptMSAConfig Command-Line Utility" in the *Oracle CEP Administrator's Guide*.

   After you run the command, the value of the `password` element will be encrypted.

7. Update your adapter Java code to access the login credentials properties you have just configured and decrypt the password.

   See Section 8.10.1, "How to Pass Static Login Credentials from an Adapter to the Data Feed Provider."

8. Edit the `MANIFEST.MF` file of the application and add the `com.bea.core.encryption` package to the `Import-Package` header. See Section 14.2.2.1, "Creating the MANIFEST.MF File."

9. Re-assemble and deploy your application as usual. See Chapter 14, "Assembling and Deploying Oracle CEP Applications."

8.10.3 Updating the Adapter Code to Access the Login Credential Properties

This section describes how update your adapter Java code to dynamically get the user and password values from the extended adapter configuration, and then use the `com.bea.core.encryption.EncryptionService` API to decrypt the encrypted password.

The code snippets below build on the HelloWorld adapter Java code, shown in Section 8.3, "Implementing an Adapter or Event Bean as an Event Source."

- Import the additional APIs that you will need to decrypt the encrypted password:

```
import com.bea.core.encryption.EncryptionService;
import com.bea.core.encryption.EncryptionServiceException;
import com.bea.wlevs.util.Service;
```

■ Use the `@Service` annotation to get a reference to the `EncryptionService`:

```
private EncryptionService encryptionService;
...
@Service
public void setEncryptionService(EncryptionService encryptionService) {
    this.encryptionService = encryptionService;
}
```

■ In the `@Prepare` callback method, get the values of the `user` and `password` properties of the extended adapter configuration as usual (only code for the `password` value is shown):

```
private String password;
...
public String getPassword() {
    return password;
}
public void setPassword(String password) {
    this.password = password;
...
@Prepare
public void checkConfiguration(HelloWorldAdapterConfig adapterConfig) {
    if (adapterConfig.getMessage() == null
            || adapterConfig.getMessage().length() == 0) {
        throw new RuntimeException("invalid message: " + message);
    }
    this.password= adapterConfig.getPassword();
    ...
}
```

See Section 8.9.3, "Programming Access to the Configuration of an Adapter or Event Bean" for information about accessing the extended adapter configuration.

■ Use the `EncryptionService.decryptStringAsCharArray()` method in the `@Prepare` callback method to decrypt the encrypted password:

```
@Prepare
public void checkConfiguration(HelloWorldAdapterConfig adapterConfig) {
    if (adapterConfig.getMessage() == null
            || adapterConfig.getMessage().length() == 0) {
        throw new RuntimeException("invalid message: " + message);
    }
    this.password= adapterConfig.getPassword();
    try {
        char[] decrypted =
encryptionService.decryptStringAsCharArray(password);
        System.out.println("DECRYPTED PASSWORD is "+ new String(decrypted));
    } catch (EncryptionServiceException e) {
        throw new RuntimeException(e);
    }
}
```

The signature of the `decryptStringAsCharArray()` method is as follows:

```
char[] decryptStringAsCharArray(String encryptedString)
                              throws EncryptionServiceException
```

- Pass these credentials to the data feed provider using the vendor API.

8.11 Assembling an Adapter or Event Bean in Its Own Bundle

This section applies to both adapters and event beans. For simplicity the text mentions only adapters.

In the procedure described in Section 8.2, "Implementing an Adapter or Event Bean," it is assumed that the adapter and adapter factory are bundled in the same application JAR file that contains the other components of the event network, such as the processor, streams, and business logic POJO.

However, you might sometimes want to bundle the adapter in its own JAR file and then reference the adapter in other application bundles. This is useful if, for example, two different applications read data coming from the same data feed provider and both applications use the same event types. In this case, it makes sense to share a single adapter and event type implementations rather than duplicate the implementation in two different applications.

There is no real difference in *how* you configure an adapter and an application that uses it in separate bundles; the difference lies in *where* you put the configuration, as described in the following guidelines:

- Create an OSGI bundle that contains only the adapter Java class, the adapter factory Java class, and optionally, the event type Java class into which the adapter converts incoming data. For simplicity, it is assumed that this bundle is called `GlobalAdapter`.

- In the EPN assembly file of the `GlobalAdapter` bundle:

- Register the adapter factory as an OSGI service as usual, as described in Section 8.7.1, "Registering the Adapter or Event Bean Factory."

- If you are also including the event type in the bundle, register it as described in Section 1.5, "Creating Oracle CEP Event Types."

- Do *not* declare the adapter component using the `wlevs:adapter` element. You will use this tag in the EPN assembly file of the application bundle that actually uses the adapter.

- If you want to further configure the adapter, follow the usual procedure as described in Section 8.8, "Configuring an Adapter or Event Bean."

- If you are including the event type in the `GlobalAdapter` bundle, export the JavaBean class in the `MANIFEST.MF` file of the `GlobalAdapter` bundle. Use the `Export-Package` header, as described in Section 14.2.2.1, "Creating the MANIFEST.MF File."

- Assemble and deploy the `GlobalAdapter` bundle in the usual way, as described in Chapter 14, "Assembling and Deploying Oracle CEP Applications."

- In the EPN assembly file of the application that is going to use the adapter, declare the adapter component in the usual way, as described in Section 8.7.2, "Declaring the Adapter and Event Bean Components in your Application." You still use the `provider` attribute to specify the OSGI-registered adapter factory, although in this case the OSGI registration happens in a different EPN assembly file (of the `GlobalAdapter` bundle) from the EPN assembly file that actually uses the adapter.

- If you have exported the event type in the `GlobalAdapter` bundle, you must explicitly import it into the application that is going to use it. You do this by

adding the package to the `Import-Package` header of the MANIFEST.MF file of the application bundle, as described in Section 14.2.2.1, "Creating the MANIFEST.MF File."

9

Configuring Channels

This section contains information on the following subjects:

- Section 9.1, "Overview of Channel Configuration"
- Section 9.2, "Configuring a Channel"
- Section 9.3, "Example Channel Configuration Files"

9.1 Overview of Channel Configuration

An Oracle CEP application contains one or more channel components. A channel represents the physical conduit through which events flow between other types of components, such as between adapters and processors, and between processors and event beans (business logic POJOs). Although channels are optional between adapters and beans, they must be present up-stream and down-stream of all Oracle CQL processors.

A channel may be used with both streams and relations. For more information, see Section 1.1.4, "Stream Sources and Stream Sinks and Relation Sources and Relation Sinks".

When you create a channel in your Event Processing Network (EPN), it has a default configuration. For complete details, see Section D.10, "wlevs:channel".

The default channel configuration is typically adequate for most applications.

However, if you want to change this configuration, you must create a `channel` element in a component configuration file. In this `channel` element, you can specify channel configuration that overrides the defaults.

The component configuration file `channel` element's `name` element must match the EPN assembly file `channel` element's `id` attribute. For example, given the EPN assembly file `channel` element shown in Example 9–1, the corresponding component configuration file `channel` element is shown in Example 9–2.

Example 9–1 EPN Assembly File Channel Id: priceStream

```
<wlevs:channel id="priceStream" event-type="PriceEvent">
    <wlevs:listener ref="filterFanoutProcessor" />
    <wlevs:source ref="PriceAdapter" />
</wlevs:channel>
```

Example 9–2 Component Configuration File Channel Name: priceStream

```
<channel>
    <name>priceStream</name>
```

```
    <max-size>10000</max-size>
    <max-threads>4</max-threads>
</channel>
```

You can create a `channel` element in any of the following component configuration files:

- The default Oracle CEP application configuration file (by default, `META-INF/wlevs/config.xml`).

- A separate configuration file.

If your application has more than one channel, you can create a channel element for each of them in the default `config.xml` file, you can create separate XML files in `META-INF/wlevs` for each, or create a single XML file in `META-INF/wlevs` that contains the configuration for all channels, or even all components of your application (adapters, processors, and channels). Choose the method that best suits your development environment.

By default, Oracle CEP IDE for Eclipse creates one component configuration file and one EPN assembly file.

Component configuration files are deployed as part of the Oracle CEP application bundle. You can later update this configuration at runtime using Oracle CEP Visualizer, the `wlevs.Admin` utility, or manipulating the appropriate JMX Mbeans directly.

For more information, see:

- Section 9.1.1, "System-Timestamped Channels"

- Section 9.1.2, "Application-Timestamped Channels"

- Section 9.1.3, "Channels Representing Streams and Relations"

- Section 1.1.5, "Component Configuration Files"

- Section 1.4, "Creating the EPN Assembly File"

- *Oracle CEP Visualizer User's Guide*

- "wlevs.Admin Command-Line Reference" in the *Oracle CEP Administrator's Guide*

- "Configuring JMX for Oracle CEP" in the *Oracle CEP Administrator's Guide*

9.1.1 System-Timestamped Channels

By default, channels are system-timestamped. In this case, Oracle CEP will assign a new time from the CPU clock under two conditions: when a new event arrives, and when the configurable heartbeat timeout expires.

For more information, see:

- Section 9.2.1, "How to Configure a System-Timestamped Channel Using Oracle CEP IDE for Eclipse"

- Section C.41, "heartbeat-timeout"

9.1.2 Application-Timestamped Channels

Optionally, you can configure a channel to be application-timestamped. In this case, the time-stamp of an event is determined by the configureable `wlevs:expression` tag. A common example of an expression is a reference to a property on the event. If no expression is specified, then the time-stamp may be propagated from a prior event.

For example, this is the case when you have a system-timestamped channel from one Oracle CQL processor feeding events into an application-timestamped channel of another downstream Oracle CQL processor.

In addition, an application can use `StreamSender.sendHeartbeat()` to send an event of type `heart-beat` downstream to `StreamSink` listeners in the EPN.

For more information, see:

- Section 9.2.2, "How to Configure an Application-Timestamped Channel Using Oracle CEP IDE for Eclipse"

- Section D.3, "wlevs:application-timestamped"

- Section D.14, "wlevs:expression"

9.1.3 Channels Representing Streams and Relations

A channel can represent either a stream or a relation:

- A stream supports appends only. You specify a channel as a stream by setting EPN assembly element `wlevs:channel` attribute `is-relation` to `false` (the default).

- A relation supports inserts, deletes, and updates. You specify a channel as a realtion by setting EPN assembly element `wlevs:channel` attribute `is-relation` to `true`.

For more information, see Section 1.1.4, "Stream Sources and Stream Sinks and Relation Sources and Relation Sinks".

9.2 Configuring a Channel

You can configure a channel manually or by using the Oracle CEP IDE for Eclipse.

See Section B.1, "Component Configuration Schema wlevs_application_config.xsd" for the complete XSD Schema that describes the channel component configuration file.

See Section 9.3, "Example Channel Configuration Files" for a complete example of a channel configuration file.

This section describes the following topics:

- Section 9.2.1, "How to Configure a System-Timestamped Channel Using Oracle CEP IDE for Eclipse"

- Section 9.2.2, "How to Configure an Application-Timestamped Channel Using Oracle CEP IDE for Eclipse"

- Section 9.2.3, "How to Create a Channel Component Configuration File Manually"

9.2.1 How to Configure a System-Timestamped Channel Using Oracle CEP IDE for Eclipse

This section describes how to create a system-timestamped channel.

The most efficient and least error-prone way to create and edit a channel configuration in the default component configuration file is to use the Oracle CEP IDE for Eclipse. Optionally, you can create a channel configuration file manually (see Section 9.2.3, "How to Create a Channel Component Configuration File Manually").

For more information, see:

- Section 9.1.1, "System-Timestamped Channels"

- Chapter 5, "Oracle CEP IDE for Eclipse and the Event Processing Network"

To configure a channel using Oracle CEP IDE for Eclipse:

1. Use Oracle CEP IDE for Eclipse to create a channel.

   See Section 5.4.1.1, "How to Create a Basic Node".

2. Optionally, override the default channel assembly file configuration by adding additional `wlevs:channel` attributes and child elements:

   a. Right-click the channel node and select **Go To Assembly Source**.

   b. Add the appropriate `wlevs:channel` attributes.

   In particular, specify whether this channel is a stream or relation by configuring attribute `is-relation`:

   – To specify this channel as stream , set `is-relation` to `false` (default).

   – To specify this channel as a relation, set `is-relation` to `true`.

   See Table D–9, " Attributes of the wlevs:channel Application Assembly Element".

   c. Add the appropriate `wlevs:channel` child elements.

   – Appendix D.17, "wlevs:instance-property"

   – Appendix D.18, "wlevs:listener"

   – Appendix D.21, "wlevs:property"

   – Appendix D.22, "wlevs:source"

3. In the Project Explorer, expand your `META-INF/wlevs` directory.

4. Choose the component configuration file you want to use:

   a. To use the default component configuration file, right-click the `META-INF/wlevs/config.xml` file and select **Open With > XML Editor**.

   The file opens in an XML Editor.

   b. To create a new component configuration file:

   – Right-click the `wlevs` directory and select **New > File**.

   The New File dialog appears.

   – Enter a file name.

   You can name the file anything you want but the name of the file must end in `.xml`.

   – Click **Finish**.

   Oracle CEP IDE for Eclipse adds the component configuration file to the `wlevs` directory.

5. Right-click the component configuration file you chose to use and select **Open With > XML Editor**.

   The file opens in an XML Editor.

6. If you created a new component configuration file, add the header and `config` element shown in Example 9–3. Otherwise, proceed to step 7.

Example 9–3 Component Configuration File Header and config Element

```
<?xml version="1.0" encoding="UTF-8"?>
<n1:config xsi:schemaLocation="http://www.bea.com/ns/wlevs/config/application
wlevs_application_config.xsd"
xmlns:n1="http://www.bea.com/ns/wlevs/config/application"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">

</config>
```

7. Add a `channel` element for the channel as Example 9–4 shows.

Example 9–4 Component Configuration File Channel Element

```
<?xml version="1.0" encoding="UTF-8"?>
<n1:config xsi:schemaLocation="http://www.bea.com/ns/wlevs/config/application
wlevs_application_config.xsd"
xmlns:n1="http://www.bea.com/ns/wlevs/config/application"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
    <processor>
        ...
    </processor>
    ...
    <channel>
    </channel>
</config>
```

8. Add a `name` child element to the `channel` element.

The `name` element value must match the corresponding EPN assembly file `channel` element's `id` attribute.

For example, given the EPN assembly file `channel` element shown in Example 9–5, the corresponding configuration file `channel` element is shown in Example 9–6.

Example 9–5 EPN Assembly File Channel Id: priceStream

```
<wlevs:channel id="priceStream" event-type="PriceEvent">
    <wlevs:listener ref="filterFanoutProcessor" />
    <wlevs:source ref="PriceAdapter" />
</wlevs:channel>
```

Example 9–6 Component Configuration File Channel Name: priceStream

```
<channel>
    <name>priceStream</name>
</channel>
```

> **Caution:** Identifiers and names in XML files are case sensitive. Be sure to specify the same case when referencing the component's identifier in the EPN assembly file.

9. Optionally, override the default channel configuration by adding additional `channel` child elements:

■ Add a `max-threads` child element to specify the maximum number of threads that Oracle CEP server uses to process events for this channel.

Setting this value has no effect when `max-size` is 0. The default value is 0.

```
<channel>
    <name>priceStream</name>
    <max-threads>2</size>
</channel>
```

When set to 0, the channel acts as a pass-through. When `max-threads > 0`, the channel acts as classic blocking queue, where upstream components are producers of events and the downstream components are the consumers of events. The queue size is defined by the configuration `max-size`. There will be up to `max-threads` number of threads consuming events from the queue.

- Add a `max-size` child element to specify the maximum size of the channel.

  Zero-size channels synchronously pass-through events.

  Non-zero size channels process events asynchronously, buffering events by the requested size. The default value is 0.

  ```
  <channel>
      <name>priceStream</name>
      <max-size>10000</size>
  </channel>
  ```

- Add a `heartbeat-timeout` child element to specify the number of nanoseconds a channel can be idle before Oracle CEP generates a heartbeat event to advance time.

  The heartbeat child element applies to system-timestamped relations or streams only when no events arrive in the event channels that are feeding the processors and the processor has been configured with a statement that includes some temporal operator, such as a time-based window or a pattern matching with duration.

  ```
  <channel>
      <name>MatchOutputChannel</name>
      <heartbeat-timeout>10000</heartbeat-timeout>
  </channel>
  ```

- Add a `selector` child element to specify which up-stream Oracle CQL processor queries are permitted to output their results to the channel.

  Figure 9–1 shows an EPN with channel `filteredStream` connected to an up-stream Oracle CQL processor `filteredFanoutProcessor`.

Figure 9–1   EPN With Oracle CQL Processor and Down-Stream Channel

Example 9–7 shows the queries configured for the Oracle CQL processor.

Example 9–7   filterFanoutProcessor Oracle CQL Queries

```
<processor>
    <name>filterFanoutProcessor</name>
    <rules>
        <query id="Yr3Sector"><![CDATA[
            select cusip, bid, srcId, bidQty, ask, askQty, seq
            from priceStream where sector="3_YEAR"
        ]]></query>
```

```
        <query id="Yr2Sector"><![CDATA[
            select cusip, bid, srcId, bidQty, ask, askQty, seq
            from priceStream where sector="2_YEAR"
        ]]></query>
        <query id="Yr1Sector"><![CDATA[
            select cusip, bid, srcId, bidQty, ask, askQty, seq
            from priceStream where sector="1_YEAR"
        ]]></query>
    </rules>
</processor>
```

If you specify more than one query for an Oracle CQL processor as Example 9–7 shows, then, by default, all query results are output to the processor's out-bound channel (filteredStream in Figure 9–1). Optionally, in the component configuration source, you can use the channel element selector attribute to specify a space-delimited list of one or more Oracle CQL query names that may output their results to the channel as Example 9–8 shows. In this example, query results for query Yr3Sector and Yr2Sector are output to filteredStream but not query results for query Yr1Sector.

Example 9–8   Using selector to Control Which Query Results are Output

```
<channel>
    <name>filteredStream</name>
    <selector>Yr3Sector Yr2Sector</selector>
</channel>
```

> **Note:**   The selector attribute is only applicable if the up-stream node is an Oracle CQL processor. For more information, see Chapter 10, "Configuring Oracle CQL Processors".

For more information, see wlevs_application_config.xsd schema file.

10. Select **File > Save**.

The EPN Editor adds a configuration badge to the channel as Figure 9–2 shows. For more information, see Section 5.2.4, "Configuration Badging".

Figure 9–2   Channel With Configuration Badge
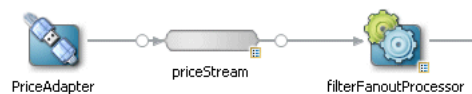
9.2.2 How to Configure an Application-Timestamped Channel Using Oracle CEP IDE for Eclipse

This section describes how to create an application-timestamped channel.

The most efficient and least error-prone way to create and edit a channel configuration in the default component configuration file is to use the Oracle CEP IDE for Eclipse. Optionally, you can create a channel configuration file manually (see Section 9.2.3, "How to Create a Channel Component Configuration File Manually").

For more information, see:

- Section 9.1.2, "Application-Timestamped Channels"

■ Chapter 5, "Oracle CEP IDE for Eclipse and the Event Processing Network"

To configure a channel using Oracle CEP IDE for Eclipse:

1. Use Oracle CEP IDE for Eclipse to create a channel.

   See Section 5.4.1.1, "How to Create a Basic Node".

2. Optionally, override the default channel assembly file configuration by adding additional `wlevs:channel` attributes and child elements:

   a. Right-click the channel node and select **Go To Assembly Source**.

   b. Add the appropriate `wlevs:channel` attributes.

   In particular, specify whether this channel is a stream or relation by configuring attribute `is-relation`:

   – To specify this channel as stream , set `is-relation` to `false` (default).

   – To specify this channel as a relation, set `is-relation` to `true`.

   See Table D–9, " Attributes of the wlevs:channel Application Assembly Element".

   c. Add a `wlevs:application-timestamped` child element.

   Use this element to specify a `wlevs:expression` child element that Oracle CEP uses to generate timestamp values.

   Optionally, configure the `wlevs:application-timestamped` attributes:

   – `is-total-order`: specifies if the application time published is always strictly greater than the last value used.

   Valid values are `true` or `false`. Default: `false`.

   For more information, see Appendix D.3, "wlevs:application-timestamped".

   d. Add other appropriate `wlevs:channel` child elements.

   – Appendix D.17, "wlevs:instance-property"

   – Appendix D.18, "wlevs:listener"

   – Appendix D.21, "wlevs:property"

   – Appendix D.22, "wlevs:source"

3. In the Project Explorer, expand your `META-INF/wlevs` directory.

4. Choose the component configuration file you want to use:

   a. To use the default component configuration file, right-click the `META-INF/wlevs/config.xml` file and select **Open With > XML Editor**.

   The file opens in an XML Editor.

   b. To create a new component configuration file:

   – Right-click the `wlevs` directory and select **New > File**.

   The New File dialog appears.

   – Enter a file name.

   You can name the file anything you want but the name of the file must end in `.xml`.

   – Click **Finish**.

Oracle CEP IDE for Eclipse adds the component configuration file to the `wlevs` directory.

5. Right-click the component configuration file you chose to use and select **Open With > XML Editor**.

   The file opens in an XML Editor.

6. If you created a new component configuration file, add the header and `config` element shown in Example 9–3. Otherwise, proceed to step 7.

Example 9–9   Component Configuration File Header and config Element

```
<?xml version="1.0" encoding="UTF-8"?>
<n1:config xsi:schemaLocation="http://www.bea.com/ns/wlevs/config/application
wlevs_application_config.xsd"
xmlns:n1="http://www.bea.com/ns/wlevs/config/application"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">

</config>
```

7. Add a `channel` element for the channel as Example 9–4 shows.

Example 9–10   Component Configuration File Channel Element

```
<?xml version="1.0" encoding="UTF-8"?>
<n1:config xsi:schemaLocation="http://www.bea.com/ns/wlevs/config/application
wlevs_application_config.xsd"
xmlns:n1="http://www.bea.com/ns/wlevs/config/application"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
    <processor>
        ...
    </processor>
    ...
    <channel>
    </channel>
</config>
```

8. Add a `name` child element to the `channel` element.

   The `name` element value must match the corresponding EPN assembly file `channel` element's `id` attribute.

   For example, given the EPN assembly file `channel` element shown in Example 9–5, the corresponding configuration file `channel` element is shown in Example 9–6.

Example 9–11   EPN Assembly File Channel Id: priceStream

```
<wlevs:channel id="priceStream" event-type="PriceEvent">
    <wlevs:listener ref="filterFanoutProcessor" />
    <wlevs:source ref="PriceAdapter" />
</wlevs:channel>
```

Example 9–12   Component Configuration File Channel Name: priceStream

```
<channel>
    <name>priceStream</name>
</channel>
```

> **Caution:** Identifiers and names in XML files are case sensitive. Be sure to specify the same case when referencing the component's identifier in the EPN assembly file.

9. Optionally, override the default channel configuration by adding additional `channel` child elements:

- Add a `max-threads` child element to specify the maximum number of threads that Oracle CEP server uses to process events for this channel.

  Setting this value has no effect when `max-size` is 0. The default value is 0.

  ```
  <channel>
      <name>priceStream</name>
      <max-threads>2</size>
  </channel>
  ```

  When set to 0, the channel acts as a pass-through. When `max-threads > 0`, the channel acts as classic blocking queue, where upstream components are producers of events and the downstream components are the consumers of events. The queue size is defined by the configuration `max-size`. There will be up to `max-threads` number of threads consuming events from the queue.

- Add a `max-size` child element to specify the maximum size of the channel.

  Zero-size channels synchronously pass-through events.

  Non-zero size channels process events asynchronously, buffering events by the requested size. The default value is 0.

  ```
  <channel>
      <name>priceStream</name>
      <max-size>10000</size>
  </channel>
  ```

- Add a `selector` child element to specify which up-stream Oracle CQL processor queries are permitted to output their results to the channel.

  Figure 9–1 shows an EPN with channel `filteredStream` connected to an up-stream Oracle CQL processor `filteredFanoutProcessor`.

Figure 9–3   EPN With Oracle CQL Processor and Down-Stream Channel

Example 9–7 shows the queries configured for the Oracle CQL processor.

Example 9–13   filterFanoutProcessor Oracle CQL Queries

```
<processor>
    <name>filterFanoutProcessor</name>
    <rules>
        <query id="Yr3Sector"><![CDATA[
            select cusip, bid, srcId, bidQty, ask, askQty, seq
            from priceStream where sector="3_YEAR"
        ]]></query>
        <query id="Yr2Sector"><![CDATA[
            select cusip, bid, srcId, bidQty, ask, askQty, seq
```

```
                from priceStream where sector="2_YEAR"
            ]]></query>
            <query id="Yr1Sector"><![CDATA[
                select cusip, bid, srcId, bidQty, ask, askQty, seq
                from priceStream where sector="1_YEAR"
            ]]></query>
        </rules>
</processor>
```

If you specify more than one query for an Oracle CQL processor as Example 9–7 shows, then, by default, all query results are output to the processor's out-bound channel (`filteredStream` in Figure 9–1). Optionally, in the component configuration source, you can use the `channel` element `selector` attribute to specify a space-delimited list of one or more Oracle CQL query names that may output their results to the channel as Example 9–8 shows. In this example, query results for query `Yr3Sector` and `Yr2Sector` are output to `filteredStream` but not query results for query `Yr1Sector`.

Example 9–14   Using selector to Control Which Query Results are Output

```
<channel>
    <name>filteredStream</name>
    <selector>Yr3Sector Yr2Sector</selector>
</channel>
```

> **Note:** The `selector` attribute is only applicable if the up-stream node is an Oracle CQL processor. For more information, see Chapter 10, "Configuring Oracle CQL Processors".

For more information, see `wlevs_application_config.xsd` schema file.

10. Select **File > Save**.

   The EPN Editor adds a configuration badge to the channel as Figure 9–2 shows. For more information, see Section 5.2.4, "Configuration Badging".

Figure 9–4   Channel With Configuration Badge

9.2.3 How to Create a Channel Component Configuration File Manually

Although the Oracle CEP IDE for Eclipse is the most efficient and least error-prone way to create and a channel configuration file (see Section 9.2.1, "How to Configure a System-Timestamped Channel Using Oracle CEP IDE for Eclipse"), alternatively, you can also create and maintain a channel configuration file manually.

For simplicity, the following procedure assumes that you are going to configure all components of an application in a single XML file.

To create a channel component configuration file manually:

1. Create an EPN assembly file and add a wlevs:channel element for each channel in your application.

   Uniquely identify each wlevs:channel with the `id` attribute.

See Section 1.4, "Creating the EPN Assembly File" for details.

2. Optionally, override the default channel assembly file configuration by adding additional `wlevs:channel` attributes and child elements:

   a. Add the appropriate `wlevs:channel` attributes.

      See Table D–9, " Attributes of the wlevs:channel Application Assembly Element".

   b. Add the appropriate `wlevs:channel` child elements.

      – Appendix D.3, "wlevs:application-timestamped"

      – Appendix D.17, "wlevs:instance-property"

      – Appendix D.18, "wlevs:listener"

      – Appendix D.21, "wlevs:property"

      – Appendix D.22, "wlevs:source"

3. Create an XML file using your favorite XML editor. You can name this XML file anything you want, provided it ends with the `.xml` extension.

The root element of the configuration file is `config`, with namespace definitions shown in the next step.

4. For each channel in your application, add a `channel` child element of `config`.

Uniquely identify each channel with the `name` child element. This name must be the same as the value of the `id` attribute in the `channel` element of the EPN assembly file that defines the event processing network of your application. This is how Oracle CEP knows to which particular channel component in the EPN assembly file this channel configuration applies. See Section 1.4, "Creating the EPN Assembly File" for details.

For example, if your application has two streams, the configuration file might initially look like:

```xml
<?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">
<helloworld:config
  xmlns:helloworld="http://www.bea.com/xml/ns/wlevs/example/helloworld">
  <processor>
  ...
  </processor>
  <channel>
    <name>firstStream</name>
    ...
  </channel>
  <channel>
    <name>secondStream</name>
    ...
  </channel>
</helloworld:config>
```

In the example, the configuration file includes two channels called `firstStream` and `secondStream`. This means that the EPN assembly file must include at least two channel registrations with the same identifiers:

```xml
<wlevs:channel id="firstStream" ...>
  ...
</wlevs:channel>
```

```
<wlevs:channel id="secondStream" ...>
 ...
</wlevs:channel>
```

> **Caution:** Identifiers and names in XML files are case sensitive, so be sure you specify the same case when referencing the component's identifier in the EPN assembly file.

5. Optionally, override the default channel configuration by adding additional `channel` child elements:

   - Add a `max-threads` child element to specify the maximum number of threads that Oracle CEP server uses to process events for this channel.

     Setting this value has no effect when `max-size` is 0. The default value is 0.

     ```
     <channel>
         <name>priceStream</name>
         <max-threads>2</size>
     </channel>
     ```

     When set to 0, the channel acts as a pass-through. When `max-threads > 0`, the channel acts as classic blocking queue, where upstream components are producers of events and the downstream components are the consumers of events. The queue size is defined by the configuration `max-size`. There will be up to `max-threads` number of threads consuming events from the queue.

   - Add a `max-size` child element to specify the maximum size of the channel.

     Zero-size channels synchronously pass-through events.

     Non-zero size channels process events asynchronously, buffering events by the requested size. The default value is 0.

     ```
     <channel>
         <name>priceStream</name>
         <max-size>10000</size>
     </channel>
     ```

   - Add a `heartbeat-timeout` child element to specify the number of nanoseconds a channel can be idle before Oracle CEP generates a heartbeat event to advance time.

     The heartbeat child element applies to system-timestamped relations or streams only when no events arrive in the event channels that are feeding the processors and the processor has been configured with a statement that includes some temporal operator, such as a time-based window or a pattern matching with duration.

     ```
     <channel>
         <name>MatchOutputChannel</name>
         <heartbeat-timeout>10000</heartbeat-timeout>
     </channel>
     ```

   - Add a `selector` child element to specify which up-stream Oracle CQL processor queries are permitted to output their results to the channel.

     Figure 9–1 shows an EPN with channel `filteredStream` connected to an up-stream Oracle CQL processor `filteredFanoutProcessor`.

Figure 9–5   EPN With Oracle CQL Processor and Down-Stream Channel

Example 9–7 shows the queries configured for the Oracle CQL processor.

Example 9–15   filterFanoutProcessor Oracle CQL Queries

```
<processor>
    <name>filterFanoutProcessor</name>
    <rules>
        <query id="Yr3Sector"><![CDATA[
            select cusip, bid, srcId, bidQty, ask, askQty, seq
            from priceStream where sector="3_YEAR"
        ]]></query>
        <query id="Yr2Sector"><![CDATA[
            select cusip, bid, srcId, bidQty, ask, askQty, seq
            from priceStream where sector="2_YEAR"
        ]]></query>
        <query id="Yr1Sector"><![CDATA[
            select cusip, bid, srcId, bidQty, ask, askQty, seq
            from priceStream where sector="1_YEAR"
        ]]></query>
    </rules>
</processor>
```

If you specify more than one query for an Oracle CQL processor as
Example 9–7 shows, then, by default, all query results are output to the
processor's out-bound channel (`filteredStream` in Figure 9–1). Optionally,
in the component configuration source, you can use the `channel` element
`selector` attribute to specify a space-delimited list of one or more Oracle
CQL query names that may output their results to the channel as Example 9–8
shows. In this example, query results for query `Yr3Sector` and `Yr2Sector`
are output to `filteredStream` but not query results for query `Yr1Sector`.

Example 9–16   Using selector to Control Which Query Results are Output

```
<channel>
    <name>filteredStream</name>
    <selector>Yr3Sector Yr2Sector</selector>
</channel>
```

Note: The `selector` attribute is only applicable if the up-stream
node is an Oracle CQL processor. For more information, see
Chapter 10, "Configuring Oracle CQL Processors".

For more information, Section B.1, "Component Configuration Schema wlevs_
application_config.xsd".

6. Save and close the configuration file.

9.3 Example Channel Configuration Files

Figure 9–6 shows part of an EPN that contains two channels: `priceStream` and
`filteredStream`. The `priceStream` channel is an in-bound channel that connects
the `PriceAdapter` event source and its `PriceEvent` events to an Oracle CQL

processor `filterFanoutProcessor`. The `filteredStream` channel is an out-bound channel that connects the Oracle CQL processor's query results (`FilteredPriceEvent` events) to down-stream components (not shown in Figure 9–6).

Figure 9–6   EPN with Two Channels

This section provides example channel configuration files, including:

9.3.1 Channel Component Configuration File

Example 9–17 shows a sample component configuration file that configures the two channels shown in Figure 9–6.

Example 9–17   Sample Channel Component Configuration File

```xml
<?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>filterFanoutProcessor</name>
        <rules>
            <query id="Yr3Sector"><![CDATA[
                select cusip, bid, srcId, bidQty, ask, askQty, seq
                from priceStream where sector="3_YEAR"
            ]]></query>
        </rules>
    </processor>
    <channel>
        <name>priceStream</name>
        <max-size>10000</max-size>
        <max-threads>4</max-threads>
    </channel>
    <channel>
        <name>filteredStream</name>
        <max-size>5000</max-size>
        <max-threads>2</max-threads>
    </channel>
</n1:config>
```

9.3.2 Channel EPN Assembly File

Example 9–18 shows a EPN assembly file that configures the two channels shown in Figure 9–6.

Example 9–18   Channel EPN Assembly File

```xml
<?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"
```

```
            xmlns:cqlx="http://www.oracle.com/schema/cqlx"
          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-v11_0_0_0.xsd">

   <wlevs:event-type-repository>
       <wlevs:event-type type-name="PriceEvent">
           <wlevs:properties>
               <wlevs:property name="cusip" type="java.lang.String" />
               <wlevs:property name="bid" type="java.lang.Double" />
               <wlevs:property name="srcId" type="java.lang.String" />
               <wlevs:property name="bidQty" type="java.lang.Integer" />
               <wlevs:property name="ask" type="java.lang.Double" />
               <wlevs:property name="askQty" type="java.lang.Integer" />
               <wlevs:property name="seq" type="java.lang.Long" />
               <wlevs:property name="sector" type="java.lang.String" />
           </wlevs:properties>
       </wlevs:event-type>
       <wlevs:event-type type-name="FilteredPriceEvent">
           <wlevs:properties>
               <wlevs:property name="cusip" type="java.lang.String" />
               <wlevs:property name="bid" type="java.lang.Double" />
               <wlevs:property name="srcId" type="java.lang.String" />
               <wlevs:property name="bidQty" type="java.lang.Integer" />
               <wlevs:property name="ask" type="java.lang.Double" />
               <wlevs:property name="askQty" type="java.lang.Integer" />
               <wlevs:property name="seq" type="java.lang.Long" />
           </wlevs:properties>
       </wlevs:event-type>
       <wlevs:event-type type-name="BidAskEvent">
           <wlevs:properties>
               <wlevs:property name="cusip" type="java.lang.String" />
               <wlevs:property name="bidseq" type="java.lang.Long" />
               <wlevs:property name="bidSrcId" type="java.lang.String" />
               <wlevs:property name="bid" type="java.lang.Double" />
               <wlevs:property name="askseq" type="java.lang.Long" />
               <wlevs:property name="askSrcId" type="java.lang.String" />
               <wlevs:property name="ask" type="java.lang.Double" />
               <wlevs:property name="bidQty" type="java.lang.Integer" />
               <wlevs:property name="askQty" type="java.lang.Integer" />
               <wlevs:property name="intermediateStrategy" type="java.lang.String" />
               <wlevs:property name="correlationId" type="java.lang.Long" />
               <wlevs:property name="priority" type="java.lang.Integer" />
           </wlevs:properties>
       </wlevs:event-type>
       <wlevs:event-type type-name="FinalOrderEvent">
           <wlevs:properties>
               <wlevs:property name="cusip" type="java.lang.String" />
               <wlevs:property name="bidseq" type="java.lang.Long" />
               <wlevs:property name="bidSrcId" type="java.lang.String" />
               <wlevs:property name="bid" type="java.lang.Double" />
               <wlevs:property name="bidQty" type="java.lang.Integer" />
               <wlevs:property name="bidSourceStrategy" type="java.lang.String" />
               <wlevs:property name="askseq" type="java.lang.Long" />
               <wlevs:property name="askSrcId" type="java.lang.String" />
               <wlevs:property name="ask" type="java.lang.Double" />
               <wlevs:property name="askQty" type="java.lang.Integer" />
               <wlevs:property name="askSourceStrategy" type="java.lang.String" />
               <wlevs:property name="correlationId" type="java.lang.Long" />
           </wlevs:properties>
       </wlevs:event-type>
   </wlevs:event-type-repository>
```

```
<!-- Assemble EPN (event processing network) -->
<wlevs:adapter advertise="true" id="PriceAdapter"
    provider="csvgen">
    <wlevs:instance-property name="port" value="9008" />
    <wlevs:instance-property name="eventTypeName"
        value="PriceEvent" />
    <wlevs:instance-property name="eventPropertyNames"
        value="srcId,sector,cusip,bid,ask,bidQty,askQty,seq" />
</wlevs:adapter>

<wlevs:channel id="priceStream" event-type="PriceEvent">
    <wlevs:listener ref="filterFanoutProcessor" />
    <wlevs:source ref="PriceAdapter" />
</wlevs:channel>

<!-- By default, CQL is used for OCEP 11.0 -->
<wlevs:processor id="filterFanoutProcessor" >
</wlevs:processor>

<wlevs:channel id="filteredStream"
    event-type="FilteredPriceEvent">
    <wlevs:listener ref="bbaProcessor" />
    <wlevs:listener ref="analyticsProcessor" />
    <wlevs:source ref="filterFanoutProcessor" />
</wlevs:channel>

<!-- Explicitly specify provider CQL -->
<wlevs:processor id="bbaProcessor" provider="cql">
    <wlevs:listener ref="bidAskBBAStream" />
</wlevs:processor>

<wlevs:processor id="analyticsProcessor">
    <wlevs:listener ref="bidAskAnalyticsStream" />
</wlevs:processor>

<wlevs:channel id="bidAskBBAStream" event-type="BidAskEvent">
    <wlevs:listener ref="selectorProcessor" />
</wlevs:channel>

<wlevs:channel id="bidAskAnalyticsStream" event-type="BidAskEvent">
    <wlevs:listener ref="selectorProcessor" />
</wlevs:channel>

<wlevs:processor id="selectorProcessor">
    <wlevs:listener ref="citipocOut" />
</wlevs:processor>

<wlevs:channel id="citipocOut" event-type="FinalOrderEvent" advertise="true">
    <wlevs:listener>
        <!-- Create business object -->
        <bean id="outputBean"
            class="com.bea.wlevs.POC.citi.OutputBean"
            autowire="byName" />
    </wlevs:listener>
</wlevs:channel>
</beans>
```

10

Configuring Oracle CQL Processors

This section contains information on the following subjects:

- Section 10.1, "Overview of Oracle CQL Processor Configuration"
- Section 10.2, "Configuring an Oracle CQL Processor"
- Section 10.3, "Configuring an Oracle CQL Processor Table Source"
- Section 10.4, "Configuring an Oracle CQL Processor Cache Source"
- Section 10.5, "Example Oracle CQL Processor Configuration Files"

Note: Oracle CQL replaces Event Processing Language (EPL) in Oracle CEP Release 11*g*R1 (11.1.1). Oracle CEP supports EPL for backwards compatibility. For more information, see Chapter 11, "Configuring EPL Processors".

10.1 Overview of Oracle CQL Processor Configuration

An Oracle CEP application contains one or more complex event processors, or *processors* for short. Each processor takes as input events from one or more adapters; these adapters in turn listen to data feeds that send a continuous stream of data from a source. The source could be anything, from a financial data feed to the Oracle CEP load generator.

The main feature of an Oracle CQL processor is its associated Oracle Continuous Query Language (Oracle CQL) rules that select a subset of the incoming events to then pass on to the component that is listening to the processor. The listening component could be another processor, or the business object POJO that typically defines the end of the event processing network, and thus does something with the events, such as publish them to a client application. For more information on Oracle CQL, see the *Oracle CEP CQL Language Reference*.

For each Oracle CQL processor in your application, you must create a `processor` element in a component configuration file. In this `processor` element you specify the initial set of Oracle CQL rules of the processor and any optional processor configuration.

You can configure additional optional Oracle CQL processor features in the Oracle CQL processor EPN assembly file.

The component configuration file `processor` element's `name` element must match the EPN assembly file `processor` element's `id` attribute. For example, given the EPN assembly file `processor` element shown in Example 10–1, the corresponding component configuration file `processor` element is shown in Example 10–2.

Example 10–1   EPN Assembly File Oracle CQL Processor Id: proc

```
<wlevs:processor id="proc">
    <wlevs:table-source ref="Stock" />
</wlevs:processor>
```

Example 10–2   Component Configuration File Oracle CQL Processor Name: proc

```
<processor>
    <name>proc</name>
    <rules>
        <query id="q1"><![CDATA[
            SELECT ExchangeStream.symbol, ExchangeStream.price, Stock.exchange
            FROM   ExchangeStream [Now], Stock
            WHERE  ExchangeStream.symbol = Stock.symbol
        ]]></query>
    </rules>
</procesor>
```

You can create a `processor` element in any of the following component configuration files:

- The default Oracle CEP application configuration file (by default, `META-INF/wlevs/config.xml`).

- A separate configuration file.

If your application has more than one processor, you can create a `processor` element for each of them in the default `config.xml` file, you can create separate XML files in `META-INF/wlevs` for each, or create a single XML file in `META-INF/wlevs` that contains the configuration for all processors, or even all components of your application (adapters, processors, and channels). Choose the method that best suits your development environment.

By default, Oracle CEP IDE for Eclipse creates one component configuration file and one EPN assembly file. When you create an Oracle CQL processor using Oracle CEP IDE for Eclipse, by default, the processor element is added to the default component configuration file `META-INF/wlevs/config.xml` file. Using Oracle CEP IDE for Eclipse, you can choose to create a new configuration file or use an existing configuration file at the time you create the Oracle CQL processor.

Component configuration files are deployed as part of the Oracle CEP application bundle. You can later update this configuration at runtime using Oracle CEP Visualizer, the `wlevs.Admin` utility, or manipulating the appropriate JMX Mbeans directly.

For more information, see:

- Section 1.1.5, "Component Configuration Files"

- Section 1.1.7, "EPN Assembly File"

- Section 1.4, "Creating the EPN Assembly File"

- *Oracle CEP Visualizer User's Guide*

- "wlevs.Admin Command-Line Reference" in the *Oracle CEP Administrator's Guide*

- "Configuring JMX for Oracle CEP" in the *Oracle CEP Administrator's Guide*

For more information on Oracle CQL processor configuration, see:

- Section 10.2, "Configuring an Oracle CQL Processor"

- Section 10.3, "Configuring an Oracle CQL Processor Table Source"

- Section 10.4, "Configuring an Oracle CQL Processor Cache Source"
- Section 10.5, "Example Oracle CQL Processor Configuration Files"

10.2 Configuring an Oracle CQL Processor

You can configure a processor manually or by using the Oracle CEP IDE for Eclipse.

See Section B.1, "Component Configuration Schema wlevs_application_config.xsd" for the complete XSD Schema that describes the processor component configuration file.

See Section 10.5, "Example Oracle CQL Processor Configuration Files" for a complete example of an Oracle CQL processor component configuration file and assembly file.

This section describes the following topics:

- Section 10.2.1, "How to Configure an Oracle CQL Processor Using Oracle CEP IDE for Eclipse"
- Section 10.2.2, "How to Create an Oracle CQL Processor Component Configuration File Manually"

10.2.1 How to Configure an Oracle CQL Processor Using Oracle CEP IDE for Eclipse

The most efficient and least error-prone way to create and edit a processor is to use the Oracle CEP IDE for Eclipse. Optionally, you can create and edit a processor manually (see Section 10.2.2, "How to Create an Oracle CQL Processor Component Configuration File Manually").

To configure an Oracle CQL processor using Oracle CEP IDE for Eclipse:

1. Use Oracle CEP IDE for Eclipse to create a processor.

   See Section 5.4.1.2, "How to Create a Processor Node".

   When you use the EPN editor to create an Oracle CQL processor, Oracle CEP IDE for Eclipse prompts you to choose either the default component configuration file or a new component configuration file. For more information, see Chapter 5, "Oracle CEP IDE for Eclipse and the Event Processing Network".

2. Right-click the processor node and select **Go to Configuration Source**.

   Oracle CEP IDE for Eclipse opens the appropriate component configuration file. The default processor component configuration is shown in Example 10–3.

   The default processor component configuration includes a `name` element and `rules` element.

   Use the `rules` element to group the child elements you create to contain the Oracle CQL statements this processor executes, including:

   - `rule`: contains Oracle CQL statements that register or create user-defined windows. The `rule` element `id` attribute must match the name of the window.

   - `view`: contains Oracle CQL view statements (the Oracle CQL equivalent of subqueries). The `view` element `id` attribute defines the name of the view.

   - `query`: contains Oracle CQL select statements. The `query` element `id` attribute defines the name of the query.

   The default processor component configuration includes a dummy `query` element with id `Query`.

Example 10–3   Default Processor Component Configuration

```
<processor>
    <name>proc</name>
    <rules>
        <query id="Query"><!-- <![CDATA[ select * from MyChannel [now] ]]> -->
        </query>
    </rules>
</processor>
```

3. Replace the dummy `query` element with the `rule`, `view`, and `query` elements you create to contain the Oracle CQL statements this processor executes.

For more information, see "Introduction to Oracle CQL Queries, Views, and Joins" in the *Oracle CEP CQL Language Reference*.

4. Select **File > Save**.

5. Optionally, configure additional Oracle CQL processor features in the assembly file:

- Section 10.3, "Configuring an Oracle CQL Processor Table Source"
- Section 10.4, "Configuring an Oracle CQL Processor Cache Source"

10.2.2 How to Create an Oracle CQL Processor Component Configuration File Manually

Although the most efficient and least error-prone way to create and edit a processor configuration is to use the Oracle CEP IDE for Eclipse (see Section 10.2.1, "How to Configure an Oracle CQL Processor Using Oracle CEP IDE for Eclipse"), alternatively, you can also create and maintain a processor configuration file manually.

This section describes the main steps to create the processor configuration file manually. For simplicity, it is assumed in the procedure that you are going to configure all processors in a single XML file, although you can also create separate files for each processor.

To create an Oracle CQL processor component configuration file manually:

1. Design the set of Oracle CQL rules that the processor executes. These rules can be as simple as selecting *all* incoming events to restricting the set based on time, property values, and so on, as shown in the following:

```
SELECT *
FROM   TradeStream [Now]
WHERE  price > 10000
```

Oracle CQL is similar in many ways to Structure Query Language (SQL), the language used to query relational database tables, although the syntax between the two differs in many ways. The other big difference is that Oracle CQL queries take another dimension into account (time), and the processor executes the Oracle CQL continually, rather than SQL queries that are static.

For more information, see "Introduction to Oracle CQL Queries, Views, and Joins" in the *Oracle CEP CQL Language Reference*.

2. Create the processor configuration XML file that will contain the Oracle CQL rules you designed in the preceding step, as well as other optional features, for each processor in your application.

You can name this XML file anything you want, provided it ends with the `.xml` extension.

The root element of the processor configuration file is config, with namespace definitions shown in the next step.

3. For each processor in your application, add a processor child element of config.

Uniquely identify each processor with the name child element. This name must be the same as the value of the id attribute in the wlevs:processor element of the EPN assembly file that defines the event processing network of your application. This is how Oracle CEP knows to which particular processor component in the EPN assembly file this processor configuration applies. See Section 1.4, "Creating the EPN Assembly File" for details.

For example, if your application has two processors, the configuration file might initially look like:

```
<?xml version="1.0" encoding="UTF-8"?>
<n1:config
xsi:schemaLocation="http://www.bea.com/xml/ns/wlevs/config/application  wlevs_
application_config.xsd"
 xmlns:n1="http://www.bea.com/xml/ns/wlevs/config/application"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <processor>
    <name>firstProcessor</name>
    ...
  </processor>
  <processor>
    <name>secondProcessor</name>
    ...
   </processor>
</n1:config>
```

In the example, the configuration file includes two processors called firstProcessor and secondProcessor. This means that the EPN assembly file must include at least two processor registrations with the same identifiers:

```
<wlevs:processor id="firstProcessor" ...>
 ...
</wlevs:processor>
<wlevs:processor id="secondProcessor" ...>
 ...
</wlevs:processor>
```

Caution: Identifiers and names in XML files are case sensitive, so be sure you specify the same case when referencing the component's identifier in the EPN assembly file.

---

4. Add a rules child element to each processor element.

Use the rules element to group the child elements you create to contain the Oracle CQL statements this processor executes, including:

- rule: contains Oracle CQL statements that register or create user-defined windows. The rule element id attribute must match the name of the window.

- view: contains Oracle CQL view statements (the Oracle CQL equivalent of subqueries). The view element id attribute defines the name of the view.

■ `query`: contains Oracle CQL select statements. The `query` element `id` attribute defines the name of the query.

Use the required `id` attribute of the `view` and `query` elements to uniquely identify each rule. Use the XML `CDATA` type to input the actual Oracle CQL rule. For example:

```
<?xml version="1.0" encoding="UTF-8"?>
<n1:config
    xsi:schemaLocation="http://www.bea.com/ns/wlevs/config/application wlevs_
application_config.xsd"
    xmlns:n1="http://www.bea.com/ns/wlevs/config/application"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
    <processor>
        <name>proc</name>
        <rules>
            <rule id="range_slide"><![CDATA[
                register window range_slide(winrange int, winslide int)
implement using "MyRangeSlideWindow"
            ]]></rule>
            <view id="lastEvents" schema="cusip bid srcId bidQty"><![CDATA[
                select mod(price)
                from filteredStream[partition by srcId, cusip rows 1]
            ]]></view>
            <query id="q1"><![CDATA[
                SELECT *
                FROM   lastEvents [Now]
                WHERE  price > 10000
            ]]></query>
        </rules>
    </processor>
</n1:config>]]></query>
```

5. Save and close the file.

6. Optionally, configure additional Oracle CQL processor features in the assembly file:

   ■ Section 10.3, "Configuring an Oracle CQL Processor Table Source"

   ■ Section 10.4, "Configuring an Oracle CQL Processor Cache Source"

10.3 Configuring an Oracle CQL Processor Table Source

You can configure an Oracle CQL processor to access a table in a relational database table as an event stream in which each row in the table is represented as a tuple.

In this section, assume that you create the table you want to access using the SQL statement that Example 10–4 shows.

Example 10–4   Table Create SQL Statement

```
create table Stock (symbol varchar(16), exchange varchar(16));
```

After configuration, you can define Oracle CQL queries that access the `Stock` table as if it was just another event stream. In the following example, the query joins one event stream `ExchangeStream` with the `Stock` table:

```
SELECT ExchangeStream.symbol, ExchangeStream.price, Stock.exchange
FROM   ExchangeStream [Now], Stock
WHERE  ExchangeStream.symbol = Stock.symbol
```

10.3.1 How to Configure an Oracle CQL Processor Table Source Using Oracle CEP IDE for Eclipse

The most efficient and least error-prone way to configure an Oracle CQL processor to access a relational database table is to use the Oracle CEP IDE for Eclipse.

To configure an Oracle CQL processor table source using Oracle CEP IDE for Eclipse:

1. Create a data source for the database that contains the table you want to use.

   Example 10–5 shows an example Oracle CEP server `config.xml` file with data source `StockDS`.

Example 10–5   Oracle CEP Server config.xml File With Data Source StockDS

```
<?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">
  <domain>
     <name>ocep_domain</name>
  </domain>

...

  <data-source>
    <name>StockDs</name>
    <connection-pool-params>
      <initial-capacity>1</initial-capacity>
      <max-capacity>10</max-capacity>
    </connection-pool-params>
    <driver-params>
      <url>jdbc:derby:</url>
      <driver-name>org.apache.derby.jdbc.EmbeddedDriver</driver-name>
      <properties>
        <element>
          <name>databaseName</name>
          <value>db</value>
        </element>
        <element>
          <name>create</name>
          <value>true</value>
        </element>
      </properties>
    </driver-params>
    <!--<data-source-params>
      <jndi-names>
        <element>StockDs</element>
      </jndi-names>
      <global-transactions-protocol>None</global-transactions-protocol>
    </data-source-params>-->
  </data-source>

...

</n1:config>
```

For more information, see "Configuring Access to a Relational Database" in the *Oracle CEP Administrator's Guide*.

2. Use Oracle CEP IDE for Eclipse to create a table node.

   See Section 5.4.1.1, "How to Create a Basic Node".

3. Use Oracle CEP IDE for Eclipse to create an Oracle CQL processor.

    See Section 5.4.1.2, "How to Create a Processor Node".

4. Connect the table node to the Oracle CQL processor node.

    See Section 5.4.2.1, "How to Connect Nodes".

    The EPN Editor adds a `wlevs:table-source` element to the target processor node that references the source table.

5. Right-click the table node in your EPN and select **Go to Assembly Source**.

    Oracle CEP IDE for Eclipse opens the EPN assembly file for this table node.

6. Edit the `table` element as Example 10–6 shows and configure the `table` element attributes as shown in Table 10–1.

Example 10–6   EPN Assembly File table Element

```
<wlevs:table id="Stock" event-type="StockEvent" data-source="StockDs" />
```

Table 10–1   EPN Assembly File table Element Attributes

Attribute	Description
`id`	The name of the table source. Subsequent references to this table source use this name.
`event-type`	The `type-name` you specified for the table `event-type` you created in step 9.
`data-source`	The `data-source` name you specified in the Oracle CEP server `config.xml` file in step 1.

7. Right-click the Oracle CQL processor node connected to the table in your EPN and select **Go to Assembly Source**.

    Oracle CEP IDE for Eclipse opens the EPN assembly file for this Oracle CQL processor.

8. Edit the Oracle CQL processor element's `table-source` child element as Example 10–7 shows.

    Set the `ref` attribute to the `id` of the `table` element you specified in step 6.

Example 10–7   EPN Assembly File table-source Element

```
<wlevs:processor id="proc">
    <wlevs:table-source ref="Stock" />
</wlevs:processor>
```

9. Edit the EPN assembly file to update the `event-type-repository` element with a new `event-type` child element for the table as Example 10–8 shows.

    Create a `property` child element for each column of the table you want to access and configure the property attributes as shown in Table 10–2.

Example 10–8   EPN Assembly File event-type element for a Table

```
<wlevs:event-type-repository>
    ...
    <wlevs:event-type type-name="StockEvent">
        <wlevs:properties>
            <wlevs:property name="symbol" type="[C" length="16" />
            <wlevs:property name="exchange" type="[C" length="16" />
        </wlevs:properties>
    </wlevs:event-type>
    ...
```

```
                </wlevs:event-type-repository>
```

Table 10–2   EPN Assembly File event-type Element Property Attributes

Attribute	Description
name	The name of the table column you want to access as specified in the SQL create table statement. You do not need to specify all columns.
type	The Oracle CEP Java type from Table 10–3 that corresponds to the column's SQL data type. In Example 10–4, the type value for column symbol is [C.
length	The column size as specified in the SQL create table statement. In Example 10–4, the length of column symbol is 16.

Table 10–3   SQL Column Types and Oracle CEP Type Equivalents

SQL Type	Oracle CEP Java Type	com.bea.wlevs.ede.api.Type	Description
ARRAY	[Ljava.lang.Object		Array, of depth 1, of java.lang.Object.
BIGINT	java.math.BigInteger	bigint	An instance of java.math.BigInteger.
BINARY	[B		Array, of depth 1, of byte.
BIT	java.lang.Boolean	boolean	An instance of java.lang.Boolean.
BLOB	[B		Array, of depth 1, of byte.
BOOLEAN	java.lang.Boolean	boolean	An instance of java.lang.Boolean.
CHAR	java.lang.Character	char	An instance of java.lang.Character.
CLOB	[B		Array, of depth 1, of byte.
DATE	java.sql.Date	timestamp	An instance of java.sql.Date.
DECIMAL	java.math.BigDecimal		An instance of java.math.BigDecimal.
DOUBLE	java.lang.Double	double	An instance of java.lang.Double
FLOAT	java.lang.Double	float	An instance of java.lang.Double
INTEGER	java.lang.Integer	int	An instance of java.lang.Integer.
JAVA_OBJECT	java.lang.Object	object	An instance of java.lang.Object.
LONGNVARCHAR	[C	char	Array, of depth 1, of char.
LONGVARBINARY	[B		Array, of depth 1, of byte.
LONGVARCHAR	[C	char	Array, of depth 1, of char.
NCHAR	[C	char	Array, of depth 1, of char.
NCLOB	[B		Array, of depth 1, of byte.
NUMERIC	java.math.BigDecimal		An instance of java.math.BigDecimal.
NVARCHAR	[C	char	Array, of depth 1, of char.
OTHER	java.lang.Object	object	An instance of java.lang.Object.
REAL	java.lang.Float	float	An instance of java.lang.Float
SMALLINT	java.lang.Integer	int	An instance of java.lang.Integer.
SQLXML	xmltype	xmltype	For more information on processing XMLTYPE data in Oracle CQL, see "SQL/XML (SQLX)" in the *Oracle CEP CQL Language Reference*.
TIME	java.sql.Time		An instance of java.sql.Time.
TIMESTAMP	java.sql.Timestamp	timestamp	An instance of java.sql.Timestamp.
TINYINT	java.lang.Integer	int	An instance of java.lang.Integer.
VARBINARY	[B		Array, of depth 1, of byte.
VARCHAR	[C	char	Array, of depth 1, of char.

For more information on creating event types, see:

- Section 1.1.2, "Event Types"

- Section 1.1.3, "Event Type Data Types"

- Section 1.5, "Creating Oracle CEP Event Types".

10. Edit the EPN assembly file to add Oracle CQL queries that use the table's
    event-type as shown in Example 10–9.

Example 10–9   Oracle CQL Query Using Table Event Type StockEvent

```
<processor>
    <name>proc</name>
    <rules>
        <query id="q1"><![CDATA[
            SELECT  ExchangeStream.symbol, ExchangeStream.price, Stock.exchange
            FROM    ExchangeStream [Now], Stock
            WHERE   ExchangeStream.symbol = Stock.symbol
        ]]></query>
    </rules>
</processor>
```

10.4 Configuring an Oracle CQL Processor Cache Source

You can configure an Oracle CQL processor to access the Oracle CEP cache.

For more information, see:

- Section 12.1, "Overview of Oracle CEP Cache Configuration"

- Section 12.9, "Accessing a Cache From an Oracle CQL User-Defined Function"

- Section 12.5, "Accessing a Cache From an Oracle CQL Statement"

10.5 Example Oracle CQL Processor Configuration Files

This section provides example Oracle CQL processor configuration files, including:

- Section 10.5.1, "Oracle CQL Processor Component Configuration File"

- Section 10.5.2, "Oracle CQL Processor EPN Assembly File"

10.5.1 Oracle CQL Processor Component Configuration File

The following example shows a component configuration file for an Oracle CQL
processor.

```
<?xml version="1.0" encoding="UTF-8"?>
<n1:config
    xsi:schemaLocation="http://www.bea.com/ns/wlevs/config/application wlevs_application_
config.xsd"
    xmlns:n1="http://www.bea.com/ns/wlevs/config/application"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
    <processor>
        <name>proc</name>
        <rules>
            <view id="lastEvents" schema="cusip bid srcId bidQty ask askQty seq"><![CDATA[
                select mod(price)
                from filteredStream[partition by srcId, cusip rows 1]
            ]]></view>
            <query id="q1"><![CDATA[
                SELECT *
```

```
                    FROM   lastEvents [Now]
                    WHERE  price > 10000
                ]]></query>
            </rules>
        </processor>
</n1:config>
```

In the example, the `name` element specifies that the processor for which the Oracle CQL rules are being configured is called `proc`. This in turn implies that the EPN assembly file that defines your application must include a corresponding `wlevs:processor` element with an `id` attribute value of `proc` to link these Oracle CQL rules with an actual `proc` processor instance (see Section 10.5.2, "Oracle CQL Processor EPN Assembly File").

This Oracle CQL processor component configuration file also defines a view element to specify an Oracle CQL `view` statement (the Oracle CQL equivalent of a subquery). The results of the view's select are not output to a down-stream channel.

Finally, this Oracle CQL processor component configuration file defines a `query` element to specify an Oracle CQL query statement. The query statement selects from the view. By default, the results of a query are output to a down-stream channel. You can control this behavior in the channel configuration using a `selector` element. For more information, see Section 9.2.1, "How to Configure a System-Timestamped Channel Using Oracle CEP IDE for Eclipse"

10.5.2 Oracle CQL Processor EPN Assembly File

The following example shows an EPN assembly file for an Oracle CQL processor.

```xml
<?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-v11_0_0_0.xsd">

    <wlevs:event-type-repository>
        <wlevs:event-type type-name="ExchangeEvent">
            <wlevs:properties>
                <wlevs:property name="symbol" type="[C" length="16" />
                <wlevs:property name="price" type="java.lang.Double" />
            </wlevs:properties>
        </wlevs:event-type>
        <wlevs:event-type type-name="StockExchangeEvent">
            <wlevs:properties>
                <wlevs:property name="symbol" type="[C" length="16" />
                <wlevs:property name="price" type="java.lang.Double" />
                <wlevs:property name="exchange" type="[C" length="16" />
            </wlevs:properties>
        </wlevs:event-type>
        <wlevs:event-type type-name="StockEvent">
            <wlevs:properties>
                <wlevs:property name="symbol" type="[C" length="16" />
                <wlevs:property name="exchange" type="[C" length="16" />
            </wlevs:properties>
        </wlevs:event-type>
    </wlevs:event-type-repository>
```

```xml
                        <!-- Assemble EPN (event processing network) -->
                        <wlevs:adapter id="adapter" class="com.bea.wlevs.example.db.ExchangeAdapter" >
                            <wlevs:listener ref="ExchangeStream"/>
                        </wlevs:adapter>

                        <wlevs:channel id="ExchangeStream" event-type="ExchangeEvent" >
                            <wlevs:listener ref="proc"/>
                        </wlevs:channel>

                        <wlevs:table id="Stock" event-type="StockEvent" data-source="StockDs" />

                        <wlevs:processor id="proc" advertise="true" >
                            <wlevs:table-source ref="Stock" />
                        </wlevs:processor>

                        <wlevs:channel id="OutputStream" advertise="true" event-type="StockExchangeEvent" >
                            <wlevs:listener ref="bean"/>
                            <wlevs:source ref="proc"/>
                        </wlevs:channel>

                        <osgi:reference id="ds" interface="com.bea.core.datasource.DataSourceService"
                    cardinality="0..1" />

                        <!-- Create business object -->
                        <bean id="bean" class="com.bea.wlevs.example.db.OutputBean">
                            <property name="dataSourceService" ref="ds"/>
                        </bean>

                    </beans>
```

11

Configuring EPL Processors

This section contains information on the following subjects:

> **Note:** Oracle CQL replaces Event Processing Language (EPL) in
> Oracle CEP Release 11gR1 (11.1.1). Oracle CEP supports EPL for
> backwards compatibility. For more information, see Chapter 10,
> "Configuring Oracle CQL Processors".

11.1 Overview of EPL Processor Component Configuration

An Oracle CEP application contains one or more complex event processors, or
processors for short. Each processor takes as input events from one or more adapters;
these adapters in turn listen to data feeds that send a continuous stream of data from a
source. The source could be anything, from a financial data feed to the Oracle CEP
load generator.

The main feature of an EPL processor is its associated Event Processing Language
(EPL) rules that select a subset of the incoming events to then pass on to the
component that is listening to the processor. The listening component could be another
processor, or the business object POJO that typically defines the end of the event
processing network, and thus does something with the events, such as publish them to
a client application. For more information about EPL, see the *Oracle CEP EPL Language
Reference*.

For each EPL processor in your application, you must create a `processor` element in
a component configuration file. In this `processor` element you specify the initial set
of EPL rules of the processor and any optional processor configuration such as:

- JDBC datasource reference if your Oracle CEP application requires a connection to
  a relational database.

- Enable monitoring of the processor.

You can configure additional optional EPL processor features in the EPL processor
EPN assembly file.

The component configuration file `processor` element's `name` element must match
the EPN assembly file `processor` element's `id` attribute. For example, given the EPN

assembly file `processor` element shown in Example 11–1, the corresponding component configuration file `processor` element is shown in Example 11–2.

Example 11–1   EPN Assembly File EPL Processor Id: proc

```
<wlevs:processor id="proc" provider="epl" >
    <wlevs:table-source ref="Stock" />
</wlevs:processor>
```

Example 11–2   Component Configuration File EPL Processor Name: proc

```
<processor>
    <name>proc</name>
    <rules>
       <rule id="myRule"><![CDATA[
           SELECT symbol, AVG(price)
           FROM (SELECT * FROM MarketTrade WHERE blockSize > 10)
           RETAIN 100 EVENTS PARTITION BY symbol WITH LARGEST price
           GROUP BY symbol
           HAVING AVG(price) >= 100
           ORDER BY symbol
       ]]></rule>
    </rules>
</procesor>
```

> **Note:**   Because Oracle CQL replaces Event Processing Language (EPL) in Oracle CEP Release 11*g*R1 (11.1.1), the default processor provider is `cql`. To specify an EPL processor, in the EPN assembly file, you must set the `wlevs:processor` element `provider` attribute to `epl` as Example 11–1 shows. Oracle CEP supports EPL for backwards compatibility. For more information, see Chapter 10, "Configuring Oracle CQL Processors".

You can create a `processor` element in any of the following component configuration files:

- The default Oracle CEP application configuration file (by default, `META-INF/wlevs/config.xml`).

- A separate configuration file.

If your application has more than one processor, you can create a `processor` element for each of them in the default `config.xml` file, you can create separate XML files in `META-INF/wlevs` for each, or create a single XML file in `META-INF/wlevs` that contains the configuration for all processors, or even all components of your application (adapters, processors, and channels). Choose the method that best suits your development environment.

By default, Oracle CEP IDE for Eclipse creates one component configuration file and one EPN assembly file. When you create an EPL processor using Oracle CEP IDE for Eclipse, by default, the processor element is added to the default component configuration file `META-INF/wlevs/config.xml` file.

Component configuration files are deployed as part of the Oracle CEP application bundle. You can later update this configuration at runtime using Oracle CEP Visualizer, the `wlevs.Admin` utility, or manipulating the appropriate JMX Mbeans directly.

For more information, see:

- Section 1.1.5, "Component Configuration Files"
- Section 1.1.7, "EPN Assembly File"
- Section 1.4, "Creating the EPN Assembly File"
- *Oracle CEP Visualizer User's Guide*
- "wlevs.Admin Command-Line Reference" in the *Oracle CEP Administrator's Guide*
- "Configuring JMX for Oracle CEP" in the *Oracle CEP Administrator's Guide*

For more information on EPL processor configuration, see:

- Section 11.2, "Configuring an EPL Processor"
- Section 11.3, "Configuring an EPL Processor Cache Source"
- Section 11.4, "Example EPL Processor Configuration Files"

11.2 Configuring an EPL Processor

This section describes the main steps to create the processor configuration file. For simplicity, it is assumed in the procedure that you are going to configure all processors in a single XML file, although you can also create separate files for each processor.

See Section B.1, "Component Configuration Schema wlevs_application_config.xsd" for the complete XSD Schema that describes the processor configuration file.

See Section 11.4, "Example EPL Processor Configuration Files" for a complete example of a processor configuration file.

11.2.1 How to Configure an EPL Processor Manually

You can configure an EPL processor manually using your preferred text editor.

To configure an EPL processor:

1. Design the set of EPL rules that the processor executes. These rules can be as simple as selecting *all* incoming events to restricting the set based on time, property values, and so on, as shown in the following two examples:

```
SELECT * from Withdrawal RETAIN ALL
SELECT symbol, AVG(price)
FROM (SELECT * FROM MarketTrade WHERE blockSize > 10)
RETAIN 100 EVENTS PARTITION BY symbol WITH LARGEST price
GROUP BY symbol
HAVING AVG(price) >= 100
ORDER BY symbol
```

   EPL is similar in many ways to Structure Query Language (SQL), the language used to query relational database tables, although the syntax between the two differs in many ways. The other big difference is that EPL queries take another dimension into account (time), and the processor executes the EPL continually, rather than SQL queries that are static.

   For additional conceptual information about EPL, and examples and reference information to help you design and write your own EPL rules, see *Oracle CEP EPL Language Reference*.

2. Create the processor configuration XML file that will contain the EPL rules you designed in the preceding step, as well as other optional features, for each processor in your application.

You can name this XML file anything you want, provided it ends with the `.xml` extension.

The root element of the processor configuration file is `config`, with namespace definitions shown in the next step.

3. For each processor in your application, add a `processor` child element of `config`.

Uniquely identify each processor with the `name` child element. This name must be the same as the value of the `id` attribute in the `wlevs:processor` element of the EPN assembly file that defines the event processing network of your application. This is how Oracle CEP knows to which particular processor component in the EPN assembly file this processor configuration applies. See Section 1.4, "Creating the EPN Assembly File" for details.

For example, if your application has two processors, the configuration file might initially look like:

```
<?xml version="1.0" encoding="UTF-8"?>
<n1:config
xsi:schemaLocation="http://www.bea.com/xml/ns/wlevs/config/application  wlevs_
application_config.xsd"
xmlns:n1="http://www.bea.com/xml/ns/wlevs/config/application"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <processor>
    <name>firstProcessor</name>
    ...
  </processor>
  <processor>
    <name>secondProcessor</name>
    ...
   </processor>
</n1:config>
```

In the example, the configuration file includes two processors called `firstProcessor` and `secondProcessor`. This means that the EPN assembly file must include at least two processor registrations with the same identifiers:

```
<wlevs:processor id="firstProcessor" provider="epl"...>
  ...
</wlevs:processor>
<wlevs:processor id="secondProcessor" provider="epl"...>
  ...
</wlevs:processor>
```

> **Note:** Because Oracle CQL replaces Event Processing Language (EPL) in Oracle CEP Release 11*g*R1 (11.1.1), the default processor provider is `cql`. To specify an EPL processor, in the EPN assembly file, you must set the `wlevs:processor` element `provider` attribute to `epl`. Oracle CEP supports EPL for backwards compatibility. For more information, see Chapter 10, "Configuring Oracle CQL Processors".

> **Caution:** Identifiers and names in XML files are case sensitive, so be sure you specify the same case when referencing the component's identifier in the EPN assembly file.

4. Add a `rules` child element to each `processor` to group together one or more `rule` elements that correspond to the set of EPL rules you have designed for this processor.

   Use the required `id` attribute of the `rule` element to uniquely identify each rule. Use the XML `CDATA` type to input the actual EPL rule. For example:

   ```
   <processor>
       <name>firstProcessor</name>
       <rules>
         <rule id="myFirstRule"><![CDATA[
         SELECT * from Withdrawal RETAIN ALL
         ]]></rule>
         <rule id="mySecondRule"><![CDATA[
         SELECT * from Checking RETAIN ALL
         ]]></rule>
       </rules>
   </processor>
   ```

5. Optionally, override the default processor configuration by adding additional `processor` child elements:

   - Optionally add a `database` child element of the `processor` element to define a JDBC data source for your application. This is required if your EPL rules join a stream of events with an actual relational database table.

     Use the `name` child element of `database` to uniquely identify the datasource.

     Use the `data-source-name` child element of `database` to specify the actual name of the data source; this name corresponds to the `name` child element of the `data-source` configuration object in the `config.xml` file of your domain.

     For more information, see "Configuring Access to a Relational Database" in the *Oracle CEP Administrator's Guide*.

     For example:

     ```
     <processor>
         <name>firstProcessor</name>
         <rules>
         ....
         </rules>
         <database>
           <name>myDataSource</name>
           <data-source-name>rdbmsDataSource</data-source-name>
         </database>
     </processor>
     ```

6. Save and close the file.

7. Optionally, configure additional EPL processor features in the assembly file:

   - Section 11.3, "Configuring an EPL Processor Cache Source"

11.3 Configuring an EPL Processor Cache Source

You can configure an EPL processor to access the Oracle CEP cache.

For more information, see:

- Section 12.1, "Overview of Oracle CEP Cache Configuration"

11.4 Example EPL Processor Configuration Files

This section provides example Oracle CQL processor configuration files, including:

11.4.1 EPL Processor Component Configuration File

The following example shows how to configure one of the sample EPL queries shown in Section 11.2, "Configuring an EPL Processor" for the `myProcessor` EPL processor:

```xml
<?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>myProcessor</name>
        <rules>
            <rule id="myRule"><![CDATA[
                SELECT symbol, AVG(price)
                FROM (SELECT * FROM MarketTrade WHERE blockSize > 10)
                RETAIN 100 EVENTS PARTITION BY symbol WITH LARGEST price
                GROUP BY symbol
                HAVING AVG(price) >= 100
                ORDER BY symbol
            ]]></rule>
        </rules>
    </processor>
</n1:config>
```

In the example, the `name` element specifies that the processor for which the single EPL rule is being configured is called `myProcessor`. This in turn implies that the EPN assembly file that defines your application must include a corresponding `<wlevs:processor id="myProcessor" provider="epl" />` tag to link these EPL rules with an actual `myProcessor` EPL processor instance (see Section 11.4.2, "EPL Processor EPN Assembly File").

11.4.2 EPL Processor EPN Assembly File

The following example shows an EPN assembly file for an EPL processor.

```xml
<?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-v11_0_0.xsd">

    <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>

    <!-- Adapter can be created from a local class, without having to go through a adapter
factory -->
    <wlevs:adapter id="helloworldAdapter"
class="com.bea.wlevs.adapter.example.helloworld.HelloWorldAdapter" >
        <wlevs:instance-property name="message" value="HelloWorld - the current time is:"/>
    </wlevs:adapter>

    <wlevs:channel id="helloworldInputChannel" event-type="HelloWorldEvent" >
        <wlevs:listener ref="helloworldProcessor"/>
        <wlevs:source ref="helloworldAdapter"/>
    </wlevs:channel>

    <!-- The default processor for OCEP 11.0.0.0 is CQL -->
    <wlevs:processor id="helloworldProcessor" provider="epl" />

    <wlevs:channel id="helloworldOutputChannel" event-type="HelloWorldEvent"
advertise="true">
        <wlevs:listener>
            <bean class="com.bea.wlevs.example.helloworld.HelloWorldBean"/>
        </wlevs:listener>
        <wlevs:source ref="helloworldProcessor"/>
    </wlevs:channel>

</beans>
```

12

Configuring Caching

This section contains information on the following subjects:

12.1 Overview of Oracle CEP Cache Configuration

A *cache* is a temporary storage area for events, created exclusively to improve the overall performance of your Oracle CEP application; it is not necessary for the application to function correctly. Oracle CEP applications can optionally publish or consume events to and from a cache to increase the availability of the events and increase the performance of their applications.

A *caching system* refers to a configured instance of a caching implementation. A caching system defines a named set of configured caches as well as the configuration for remote communication if any of the caches are distributed across multiple machines.

Oracle CEP supports the following caching systems:

- Oracle CEP local cache: a local, in-memory single-JVM cache.

- Oracle Coherence: a JCache-compliant in-memory distributed data grid solution for clustered applications and application servers. It coordinates updates to the data using cluster-wide concurrency control, replicates data modifications across the cluster using the highest performing clustered protocol available, and delivers notifications of data modifications to any servers that request them. You take advantage of Oracle Coherence features using the standard Java collections API to access and modify data, and use the standard JavaBean event model to receive data change notifications.

> **Note:** Before you can use Oracle CEP with Oracle Coherence, you
> must obtain a valid Oracle Coherence license such as a license for
> Coherence Enterprise Edition, Coherence Grid Edition, or Oracle
> WebLogic Application Grid. For more information on Oracle
> Coherence, see
> http://www.oracle.com/technology/products/coherence/
> index.html.

- Third-party caches: you can create a plug-in to allow Oracle CEP to work with other, third-party cache implementations.

A cache can be considered a stage in the event processing network in which an external element (the cache) consumes or produces events; this is similar to an adapter that uses a JMS destination. A cache, however, does not have to be an actual stage in the network; another component or Spring bean can access a cache programmatically using the caching APIs.

A caching system is always configured at the application level. Other Oracle CEP applications in separate bundles can also use the caching system.

For each caching system in your application, you must create a `caching-system` element in a component configuration file. In this `caching-system` element you specify the caching system name and one or more `cache` elements. In the `cache` elements, you define optional cache configuration such as:

- Maximum cache size

- Eviction policy

- Time-to-live

- Idle time

- Write policy (write-none, write-through, or write-behind)

- Work manager

The component configuration file `caching-system` element's `name` element must match the EPN assembly file `caching-system` element's `id` attribute. For example, given the EPN assembly file `caching-system` element shown in Example 12–1, the corresponding component configuration file `caching-system` element is shown in Example 12–2.

Example 12–1   EPN Assembly File Caching System Id: cacheSystem

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

<wlevs:cache id="cache1">
    <wlevs:caching-system ref="cacheSystem"/>
</wlevs:cache>
```

Example 12–2   Component Configuration File Caching System Name: cacheSystem

```
<caching-system>
    <name>cacheSystem</name>
    <cache>
        <name>cache1</name>
        ...
    </cache>
```

```
</caching-system>
```

Similarly, the component configuration file `cache` element's `name` element must match the EPN assembly file `cache` element's `id` attribute. For example, given the EPN assembly file `cache` element shown in Example 12–1, the corresponding component configuration file `cache` element is shown in Example 12–2.

Example 12–3   EPN Assembly File Caching Id: cache1

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

<wlevs:cache id="cache1">
    <wlevs:caching-system ref="cacheSystem"/>
</wlevs:cache>
```

Example 12–4   Component Configuration File Caching Name: cache1

```
<caching-system>
    <name>cacheSystem</name>
    <cache>
        <name>cache1</name>
        ...
    </cache>
</caching-system>
```

You can create a `caching-system` element in any of the following component configuration files:

- The default Oracle CEP application configuration file (by default, `META-INF/wlevs/config.xml`).

- A separate configuration file.

If your application has more than one caching system, you can create a `caching-system` element for each of them in the default `config.xml` file, you can create separate XML files in `META-INF/wlevs` for each, or create a single XML file in `META-INF/wlevs` that contains the configuration for all caching systems, or even all components of your application (adapters, processors, and channels). Choose the method that best suits your development environment.

By default, Oracle CEP IDE for Eclipse creates one component configuration file and one EPN assembly file. When you add a cache to the EPN using Oracle CEP IDE for Eclipse, it adds a cache element to the EPN assembly file. You must manually add `caching-system` elements to the EPN assembly file and component configuration file.

Component configuration files are deployed as part of the Oracle CEP application bundle. You can later update this configuration at runtime using Oracle CEP Visualizer, the `wlevs.Admin` utility, or manipulating the appropriate JMX Mbeans directly.

For more information, see:

- Section 1.1.5, "Component Configuration Files"

- Section 1.1.7, "EPN Assembly File"

- Section 1.4, "Creating the EPN Assembly File"

- *Oracle CEP Visualizer User's Guide*

- "wlevs.Admin Command-Line Reference" in the *Oracle CEP Administrator's Guide*

- "Configuring JMX for Oracle CEP" in the *Oracle CEP Administrator's Guide*

For more information on configuring caching, see:

- Section 12.1.1, "Caching Use Cases"

- Section 12.1.2, "Additional Caching Features"

- Section 12.1.3, "Caching APIs"

12.1.1 Caching Use Cases

The following sections describe the principal Oracle CEP caching use cases.

12.1.1.1 Use Case: Publishing Events to a Cache

An example of this use case is a financial application that publishes events to a cache while the financial market is open and then processes data in the cache after the market closes.

Publishing events to a cache makes them highly available or available to other Oracle CEP applications running in the server. Publishing events to a cache also allows for asynchronous writes to a secondary storage by the cache implementation. You can configure any stage in an Oracle CEP application that generates events (input adapter, channel, business POJO, or processor) to publish its events to the cache.

12.1.1.2 Use Case: Consuming Data From a Cache

Oracle CEP applications may sometimes need to access non-streaming data in order to do its work; caching this data can increase the performance of the application.

The standard components of an Oracle CEP application that are allowed direct programming access to a cache are input- and output-adapters and business POJOs.

Additionally, applications can access a cache from Oracle CQL or EPL, either by a user-defined function or directly from an Oracle CQL or EPL statement.

In the case of a user-defined function, programmers use Spring to inject the cache resource into the implementation of the function. For more information, see Section 1.6, "Configuring Oracle CEP Resource Access".

Applications can also query a cache directly from anOracle CQL or EPL statement that runs in a processor. In this case, the cache essentially functions as another type of stream data source to a processor so that querying a cache is very similar to querying a channel except that data is pulled from a cache.

An example of using Oracle CQL to query a cache is from a financial application that publishes orders and the trades used to execute the orders to a cache. At the end of the day when the markets are closed, the application queries the cache in order to find all the trades related to a particular order.

12.1.1.3 Use Case: Updating and Deleting Data in a Cache

An Oracle CEP application can both update and delete data in a cache when required.

For example, a financial application may need to update an order in the cache each time individual trades that fulfill the order are executed, or an order may need to be deleted if it has been cancelled. The components of an application that are allowed to consume data from a cache are also allowed to update it.

12.1.1.4 Use Case: Using a Cache in a Multi-Server Domain

If build an Oracle CEP application that uses a cache, and you plan to deploy that application in a multi-server domain, then you must use a caching-system that supports a distributed cache.

In this case, you must use either Oracle Coherence or a third-party caching system that supports a distributed cache.

For more information, see:

- "Administrating element Multi-Server Domains" in the *Oracle CEP Administrator's Guide*

- Section 12.3, "Configuring an Oracle Coherence Caching System and Cache"

- Section 12.4, "Configuring a Third-Party Caching System and Cache"

12.1.2 Additional Caching Features

In addition to the major caching features described in the preceding sections, Oracle CEP caching includes the following features:

- Pre-load a cache with data before an application is deployed.

- Periodically refresh, invalidate, and flush the data in a cache. All these tasks happen incrementally and without halting the application or causing latency spikes.

- Dynamically update a cache's configuration.

12.1.3 Caching APIs

Oracle CEP provides a number of caching APIs that you can use in your application to perform certain tasks. The APIs are in two packages:

- `com.bea.cache.jcache`—Includes the APIs used to access a cache and create cache loader, listeners, and stores.

- `com.bea.wlevs.cache.spi`—Includes the API used to access a caching system.

The creation, configuration, and wiring of the caching systems and caches is all done using the EPN assembly file and component configuration files. This means that you typically never explicitly use the `Cache` and `CachingSystem` interfaces in your application; the only reason to use them is if you have additional requirements than the standard configuration. For example: if you want to provide integration with a third-party cache provider, then you must use the `CachingSystem` interface; if you want to perform operations on a cache that are not part of the `java.util.Map` `interface`, then you can use the `Cache` interface.

If you create cache listeners, loaders, or stores for an Oracle CEP local cache, then the Spring beans you write must implement the `CacheListener`, `CacheLoader`, or `CacheStore` interfaces.

If you create cache listeners, loaders, or stores for an Oracle Coherence cache, then the Spring beans you write must implement the appropriate Oracle Coherence interfaces.

If you create cache listeners, loaders, or stores for a third-party cache, then the Spring beans you write must implement the appropriate third-party cache interfaces.

For more information, see:

- Section 12.2, "Configuring an Oracle CEP Local Caching System and Cache"

- Section 12.3, "Configuring an Oracle Coherence Caching System and Cache"
- Section 12.4, "Configuring a Third-Party Caching System and Cache"
- *Oracle CEP Java API Reference*.

12.2 Configuring an Oracle CEP Local Caching System and Cache

You can configure your application to use the Oracle CEP local caching system and cache. The Oracle CEP local caching system is appropriate if you do not plan to deploy your application to a multi-server domain. If you plan to deploy your application to a multi-server domain, consider using an Oracle Coherence cache (see Section 12.3, "Configuring an Oracle Coherence Caching System and Cache").

> **Note:** It is assumed in this section that you have already created an Oracle CEP application along with its EPN assembly file and that you want to update the application to use caching. If you have not, refer to Chapter 1, "Overview of Creating Oracle CEP Applications" for details.

To configure an Oracle CEP local caching system and cache:

1. Declare the caching system in the EPN assembly file.

   To declare a caching system that uses the Oracle CEP implementation declaratively in the EPN assembly file, use the `wlevs:caching-system` element without any additional attributes, as shown in the following example:

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

   The value of the `id` attribute must match the name specified for the caching system in the external configuration metadata.

   To allow other applications to use the caching system, specify that the caching system be advertised using the `advertise` attribute (by default set to `false`):

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

2. Declare one or more caches for this caching system in the EPN assembly file.

   After you have declared a caching system for an application, you configure one or more caches using the `wlevs:cache` element:

   ```
   <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>
   ```

   The `name` attribute is optional; specify it only if the name of the cache in the caching system is different from its ID. The `wlevs:caching-system` child element references the already-declared caching system that contains the cache. You must specify this child element only if the caching system is ambiguous: there is more than one caching system declared (either implicitly or explicitly) or if the caching system is in a different application or bundle.

   You can export both the caching system and the cache as an OSGI service using the `advertise` attribute.

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

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

If the cache is advertised, then a component in the EPN of an application in a *separate* bundle can then reference it. The following example shows how a processor in one bundle can use as a cache source the cache with ID `cache-id` located in a separate bundle (called `cacheprovider`):

```
<wlevs:processor id="myProcessor2">
    <wlevs:cache-source ref="cacheprovider:cache-id"/>
</wlevs:processor>
```

The caching system is responsible for creating the cache associated with a particular name and returning a reference to the cache. The resulting cache bean implements the `java.util.Map` interface.

3. Open your application's component configuration XML file using your favorite XML editor.

4. Update the `config` root element to add a `caching-system` child element; use the `name` child element to uniquely identify it.

   This name must match the `wlevs:caching-system` element `id` attribute you specified in the EPN assembly file in step 1. This is how Oracle CEP knows to which particular caching system in the EPN assembly file this caching configuration applies.

   For example, assume your configuration file already contains a processor and an adapter (contents removed for simplicity); then the updated file might look like the following

```
<?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>
    ...
    </processor>
    <adapter>
    ...
    </adapter
    <caching-system>
        <name>caching-system-id</name>
    </caching-system>
</n1:config>
```

5. For each cache you want to create, update the `caching-system` element to add a `cache` child element; use the `name` child element to uniquely identify it.

   This `name` must match the `wlevs:cache` element `id` attribute you specified in the EPN assembly file in step 2. This is how Oracle CEP knows to which particular cache in the EPN assembly file this configuration applies.

   The following example shows two caches in the caching system:

```
<caching-system>
    <name>caching-system-id</name>
    <cache>
        <name>cache-id</name>
        ...
    </cache>
```

```
    <cache>
        <name>second-cache-id</name>
        ...
    </cache>
</caching-system>
```

6. For each cache, optionally add the following elements that take simple data types to configure the cache:

   - `max-size`: The number of cache elements in memory after which eviction/paging occurs. The maximum cache size is 2^{31}-1 entries; default is 64.

   - `eviction-policy`: The eviction policy to use when `max-size` is reached. Supported values are: FIFO, LRU, LFU, and NRU; default value is LFU.

   - `time-to-live`: The maximum amount of time, in milliseconds, that an entry is cached. Default value is infinite.

   - `idle-time`: Amount of time, in milliseconds, after which cached entries are actively removed from the cache. Default value is infinite.

   - `work-manager-name`: The work manager to be used for all asynchronous operations. The value of this element corresponds to the `name` child element of the `work-manager` element in the server's `config.xml` configuration file.

   For example:

```
<caching-system>
    <name>caching-system-id</name>
    <cache>
        <name>cache-id</name>
        <max-size>100000</max-size>
        <eviction-policy>LRU</eviction-policy
        <time-to-live>3600</time-to-live>
    </cache>
</caching-system>
```

7. Optionally add *either* `write-through` or `write-behind` as a child element of `cache` to specify synchronous or asynchronous writes to the cache store, respectively. By default, writes to the store are synchronous (`<rite-through`) which means that as soon as an entry is created or updated the write occurs.

   If you specify the `write-behind` element, then the cache store is invoked from a separate thread after a create or update of a cache entry. Use the following optional child elements to further configure the asynchronous writes to the store:

   - `work-manager-name`: The work manager that handles asynchronous writes to the cache store. If a work manager is specified for the cache itself, this value overrides it for store operations only. The value of this element corresponds to the `name` child element of the `work-manager` element in the server's `config.xml` configuration file.

   - `batch-size`: The number of updates that are picked up from the store buffer to write back to the backing store. Default value is 1.

   - `buffer-size`: The size of the internal store buffer that temporarily holds the asynchronous updates that need to be written to the store. Default value is 100.

   - `buffer-write-attempts`: The number of attempts that the user thread makes to write to the store buffer. The user thread is the thread that creates or updates a cache entry. If all attempts by the user thread to write to the store buffer fail, it will invoke the store synchronously. Default value is 1.

- **buffer-write-timeout** : The time in milliseconds that the user thread waits before aborting an attempt to write to the store buffer. The attempt to write to the store buffer fails only in case the buffer is full. After the timeout, further attempts may be made to write to the buffer based on the value of `buffer-write-attempts`. Default value is 100.

For example:

```
<caching-system>
    <name>caching-system-id</name>
    <cache>
        <name>cache-id</name>
        <max-size>100000</max-size>
        <eviction-policy>LRU</eviction-policy
        <time-to-live>3600</time-to-live>
        <write-behind>
            <buffer-size>200</buffer-size>
            <buffer-write-attempts>2</buffer-write-attempts>
            <buffer-write-timeout>200</buffer-write-timeout>
        </write-behind>
    </cache>
</caching-system>
```

8. Optionally add a `cache` element `listeners` child element to configure the behavior of components that listen to the cache.

Use the `asynchronous` Boolean attribute to specify whether listeners should be invoked:

- asynchronously: `true`.

- synchronously: `false`, which means listeners are invoked synchronously (Default).

The `listeners` element has a single child element, `work-manager-name`, that specifies the work manager to be used for asynchronously invoking listeners. This value is ignored if synchronous invocations are enabled. If a work manager is specified for the cache itself, this value overrides it for invoking listeners only. The value of this element corresponds to the `work-manager` element `name` child element in the Oracle CEP server `config.xml` configuration file.

For example:

```
<caching-system>
    <name>caching-system-id</name>
    <cache>
        <name>cache-id</name>
        <max-size>100000</max-size>
        <eviction-policy>LRU</eviction-policy
        <time-to-live>3600</time-to-live>
        <write-behind>
            <buffer-size>200</buffer-size>
            <buffer-write-attempts>2</buffer-write-attempts>
            <buffer-write-timeout>200</buffer-write-timeout>
        </write-behind>
        <listeners asynchronous="true">
            <work-manager-name>cachingWM</work-manager-name>
        </listeners>
    </cache>
</caching-system>
```

9. Optionally, override the default cache configuration by updating the EPN assembly file with one or more additional `cache` element child elements.

   - Specify that a cache is an event sink by configuring it as a listener to another component in the event processing network.

     See Section 12.2.1, "Configuring an Oracle CEP Local Cache as an Event Listener."

   - Specify that a cache is an event source to which another component in the event processing network listens.

     See Section 12.2.2, "Configuring an Oracle CEP Local Cache as an Event Source."

   - Configure a cache loader for a cache.

     See Section 12.2.3, "Configuring an Oracle CEP Local Cache Loader."

   - Configure a cache store for a cache.

     See Section 12.2.4, "Configuring an Oracle CEP Local Cache Store."

10. Access the Oracle CEP local cache:

    – Optionally reference the Oracle CEP local cache in a query statement.

      See:

      * Section 12.5, "Accessing a Cache From an Oracle CQL Statement"

      * Section 12.6, "Accessing a Cache From an EPL Statement"

    – Optionally configure and program a custom adapter, business POJO, or Oracle CQL or EPL user-defined function to access the Oracle CEP local cache.

      The configuration is done in the EPN assembly file and the programming is done in the Java file that implements the adapter, POJO, or user-defined function.

      See:

      * Section 12.7, "Accessing a Cache From an Adapter"

      * Section 12.8, "Accessing a Cache From a Business POJO"

      * Section 12.9, "Accessing a Cache From an Oracle CQL User-Defined Function"

      * Section 12.10, "Accessing a Cache From an EPL User-Defined Function"

    – Optionally, access the Oracle CEP local cache using JMX.

      See Section 12.11, "Accessing a Cache Using JMX".

11. When you assemble your application, verify that the `META-INF/MANIFEST.MF` file includes the following import as Figure 12–1 shows:

    ```
    com.bea.wlevs.cache.spi; version ="11.1.0.0"
    ```

    If the `MANIFEST.MF` files does not include this import, update the `MANIFEST.MF` file to add this import before deploying your application.

Figure 12–1 Editing the MANIFEST.MF File

12.2.1 Configuring an Oracle CEP Local Cache as an Event Listener

An Oracle CEP local cache can be configured as an explicit listener in the event processing network in order to receive events.

For example, to specify that a cache listens to a channel, specify the `wlevs:listener` element with a reference to the cache as a child of the `wlevs:channel` element as shown below:

```
<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>
...
<wlevs:channel id="tradeStream">
    <wlevs:listener ref="cache-id"/>
</wlevs:channel>
```

As the channel sends new events to the cache, they are inserted into the cache. If a *remove event* (an old event that exits the output window) is sent by the channel, then the event is removed from the cache.

12.2.1.1 Specifying the Key Used to Index an Oracle CEP Local Cache

When you configure an Oracle CEP local cache to be a listener, events are inserted into the cache. This section describes the variety of options available to you to specify the key used to index a cache in this instance.

If you do not explicitly specify a key, the event object itself serves as both the key and value when the event is inserted into the cache. In this case, the event class must include a valid implementation of the equals and hashcode methods that take into account the values of the key properties.

See the following for ways to explicitly specify a key:

- Section 12.3.2.1.1, "Specifying a Key Property in EPN Assembly File"
- Section 12.3.2.1.2, "Using a Metadata Annotation to Specify a Key"
- Section 12.3.2.1.3, "Specifying a Composite Key"

12.2.1.1.1 Specifying a Key Property in EPN Assembly File The first option is to specify a property name for the key property when a cache is declared in the EPN assembly file using the key-properties attribute, as shown in the following example:

```
<wlevs:cache id="cache-id" key-properties="key-property-name">
    <wlevs:caching-system ref="caching-system-id"/>
</wlevs:cache>
```

In this case, all events that are inserted into the cache are required to have a property of this name at runtime, otherwise Oracle CEP throws an exception.

For example, assume the event type being inserted into the cache looks something like the following; note the key property (only relevant Java source shown):

```
public class MyEvent {
    private String key;
    public MyEvent() {
    }
    public MyEvent(String key) {
        this.key = key;
    }
    public String getKey() {
        return key;
    }
    public void setKey(String key) {
        this.key = key;
    }
 ...
}
```

The corresponding declaration in the EPN assembly file would look like the following:

```
<wlevs:cache id="cache-id" key-properties="key">
    <wlevs:caching-system ref="caching-system-id"/>
</wlevs:cache>
```

12.2.1.1.2 Using a Metadata Annotation to Specify a Key The second option is to use the metadata annotation com.bea.wlevs.ede.api.Key to annotate the event property in the Java class that implements the event type. This annotation does not have any attributes.

The following example shows how to specify that the key property of the MyEvent event type is the key; only relevant code is shown:

```
import com.bea.wlevs.ede.api.Key;
public class MyEvent {
    @Key
    private String key;
    public MyEvent() {
    }
    public MyEvent(String key) {
        this.key = key;
    }
```

```
    public String getKey() {
        return key;
    }
    public void setKey(String key) {
        this.key = key;
    }
...
}
```

12.2.1.1.3 Specifying a Composite Key The final option is to use the `key-class` attribute of the `wlevs:cache` element to specify a composite key in which multiple properties form the key. The value of the `key-class` attribute must be a JavaBean whose public fields match the fields of the event class. The matching is done according to the field name. For example:

```
<wlevs:cache id="cache-id" key-class="key-class-name">
    <wlevs:caching-system ref="caching-system-id"/>
</wlevs:cache>
```

12.2.2 Configuring an Oracle CEP Local Cache as an Event Source

An Oracle CEP local cache can be configured as a source of events to which another component in the event processing network listens. The listening component can be an adapter or a standard Spring bean. Any component that listens to a cache must implement the `com.bea.cache.jcache.CacheListener` interface. The following example shows how to configure a cache to be an event source for a Spring bean:

```
<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.

12.2.3 Configuring an Oracle CEP Local Cache Loader

An Oracle CEP local cache loader is an object that loads objects into an Oracle CEP local cache. You configure a cache loader by using the `wlevs:cache-loader` child element of the `wlevs:cache` element to specify the bean that does the loading work, as shown in the following example:

```
<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-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 does the cache loading work; it is referenced by the cache using the `ref` attribute of the `wlevs:cache-loader` child element.

You must program the `wlevs.example.MyCacheLoader` class yourself, and it must implement the `com.bea.cache.jcache.CacheLoader` interface. This interface includes the `load()` method to customize the loading of a single object into the cache; Oracle CEP calls this method when the requested object is not in the cache. The interface also includes `loadAll()` methods that you implement to customize the loading of the entire cache.

12.2.4 Configuring an Oracle CEP Local Cache Store

You can configure an Oracle CEP local cache with a custom store that is responsible for writing data from the cache to a backing store, such as a table in a database. You configure a cache store by using the `wlevs:cache-store` child element of the `wlevs:cache` element to specify the bean that does the actual storing work, as shown in the following example:

```
<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-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 does the work of writing the data from the cache to the backing store; it is referenced by the cache using the `ref` attribute of the `wlevs:cache-store` child element.

You must program the `wlevs.example.MyCacheStore` class yourself, and it must implement the `com.bea.cache.jcache.CacheStore` interface. This interface includes the `store()` method that stores the data in the backing store using the passed key; Oracle CEP calls this method when it inserts data into the cache. The interface also includes the `storeAll()` method for storing a batch of data to a backing store in the case that you have configured asynchronous writes for a cache with the `write-behind` configuration element.

12.3 Configuring an Oracle Coherence Caching System and Cache

You can configure your application to use the Oracle Coherence caching system and cache. Use this caching system if you plan to deploy your application to a multi-server domain.

Using Oracle Coherence, only the first caching-system can be configured in a server. Oracle CEP will ignore any other caching systems you might configure.

> **Note:** Before you can legally use Oracle CEP with Oracle Coherence, you must obtain a valid Coherence license such as a license for Coherence Enterprise Edition, Coherence Grid Edition, or Oracle WebLogic Application Grid. For more information on Oracle Coherence, see
> http://www.oracle.com/technology/products/coherence/index.html.

> **Note:** It is assumed in this section that you have already created an Oracle CEP application along with its EPN assembly file and that you want to update the application to use caching. If you have not, refer to Chapter 1, "Overview of Creating Oracle CEP Applications" for details.

To configure an Oracle Coherence caching system and cache:

1. Declare the Oracle Coherence caching system in the EPN assembly file.

   To declare a caching system that uses the Oracle Coherence implementation declaratively in the EPN assembly file, use the `wlevs:caching-system` element as shown in the following example:

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

   The value of the `id` attribute must match the name you specify for the caching system in the application's component configuration file (see step 3).

   To allow other applications to use the caching system, specify that the caching system be advertised using the `advertise` attribute (by default set to `false`):

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

2. Declare one or more caches for this caching system in the EPN assembly file.

   After you have declared a caching system for an application, you configure one or more caches using the `wlevs:cache` element:

   ```
   <wlevs:caching-system id="caching-system-id" provider="coherence" />
   ...
   <wlevs:cache id="myCache" advertise="false">
       <wlevs:caching-system ref="caching-system-id"/>
   </wlevs:cache>
   ```

   The `wlevs:cache` element `id` attribute is mandatory. This attribute maps to the name of a cache in the Oracle Coherence configuration files (see step 3).

   The `name` attribute is optional; specify it only if the name of the cache in the caching system is different from its ID. The `wlevs:caching-system` child element references the already-declared caching system that contains the cache. You must specify this child element only if the caching system is ambiguous: there is more than one caching system declared (either implicitly or explicitly) or if the caching system is in a different application or bundle.

   You can export both the caching system and the cache as an OSGi service using the `advertise` attribute.

   ```
   <wlevs:caching-system id="caching-system-id" provider="coherence"
   advertise="true"/>
   ...
   <wlevs:cache id="cache-id" name="alternative-cache-name" advertise="true" >
       <wlevs:caching-system ref="caching-system-id"/>
   </wlevs:cache>
   ```

   If the cache is advertised, then a component in the EPN of an application in a *separate* bundle can then reference it. The following example shows how a processor in one bundle can use as a cache source the cache with ID `cache-id` located in a separate bundle called `cacheprovider`:

```
<wlevs:processor id="myProcessor2">
    <wlevs:cache-source ref="cacheprovider:cache-id"/>
</wlevs:processor>
```

The caching system is responsible for creating the cache associated with a particular name and returning a reference to the cache. The resulting cache bean implements the `java.util.Map` interface.

3. Configure the caching system and its caches by updating the caching configuration file for the application.

   See Section 12.3.1, "Configuring the Oracle Coherence Caching System and Caches."

4. Optionally, override the default cache configuration by updating the EPN assembly file with one or more additional `cache` element child elements.

   ■ Specify that a cache is an event sink by configuring it as a listener to another component in the event processing network.

     See Section 12.3.1, "Configuring the Oracle Coherence Caching System and Caches."

   ■ Specify that a cache is an event source to which another component in the event processing network listens.

     See Section 12.3.2, "Configuring an Oracle Coherence Cache as an Event Listener."

   ■ Configure *either* a `<wlevs:cache-loader>` or a `<wlevs:cache-store>` child element of the `<wlevs:cache>` element in the EPN assembly file, but not both.

     This is because Oracle Coherence combines the loader and store into a single component. You specify a cache loader when the backing store is read-only and a cache store when the backing store is read-write

     See Section 12.3.4, "Configuring an Oracle Coherence Cache Loader or Store."

5. Access the Oracle Coherence cache:

   – Optionally reference the Oracle Coherence cache in a query statement.

     See:

     * Section 12.5, "Accessing a Cache From an Oracle CQL Statement"

     * Section 12.6, "Accessing a Cache From an EPL Statement"

   – Optionally configure and program a custom adapter, business POJO, or Oracle CQL or EPL user-defined function to access the Oracle Coherence cache.

     The configuration is done in the EPN assembly file and the programming is done in the Java file that implements the adapter, POJO, or user-defined function.

     See:

     * Section 12.7, "Accessing a Cache From an Adapter"

     * Section 12.8, "Accessing a Cache From a Business POJO"

     * Section 12.9, "Accessing a Cache From an Oracle CQL User-Defined Function"

     * Section 12.10, "Accessing a Cache From an EPL User-Defined Function"

– Optionally, access the Oracle Coherence cache using JMX.

See Section 12.11, "Accessing a Cache Using JMX".

6. When you assemble your application, verify that the META-INF/MANIFEST.MF file includes the following import as Figure 12–1 shows:

```
com.bea.wlevs.cache.spi; version ="11.1.0.0"
```

If the MANIFEST.MF files does not include this import, update the MANIFEST.MF file to add this import before deploying your application.

Figure 12–2 Editing the MANIFEST.MF File
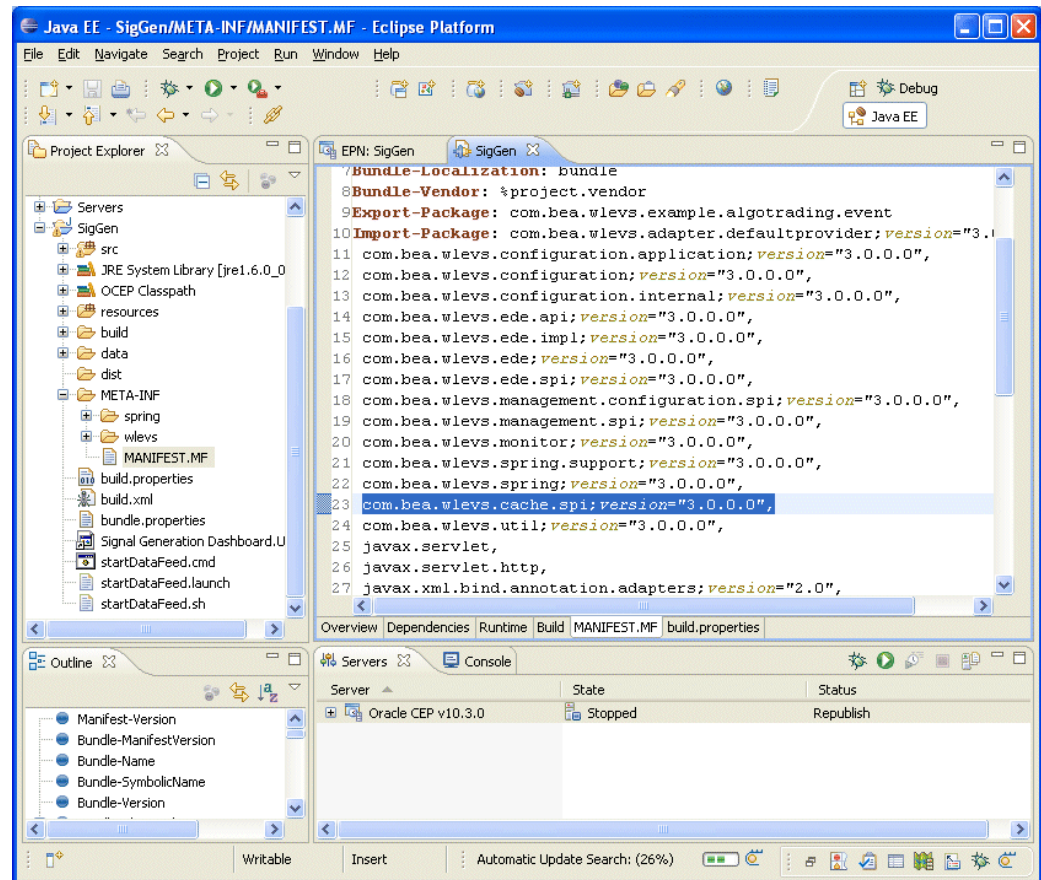
12.3.1 Configuring the Oracle Coherence Caching System and Caches

Oracle CEP leverages the native configuration provided by Oracle Coherence. You do this by packaging the following two Oracle Coherence configuration files, with the indicated names, in the application bundle that uses the Oracle Coherence cache:

■ coherence-cache-config.xml—Oracle Coherence cache configuration information. Individual caches are identified with the cache-name element; the value of this element maps to the id attribute of the wlevs:cache element in the EPN assembly file. See Section 12.3.1.1, "The coherence-cache-config.xml File" for information about this file as well as an example of the mapping.

■ tangosol-coherence-override.xml—Oracle Coherence cluster configuration. See Section 12.3.1.2, "The tangosol-coherence-override.xml File" for information about this file as well as an example.

When assembling your application, consider the following:

- `coherence-cache-config.xml` is a per-application configuration file; put this file in the `META-INF/wlevs/coherence` directory of the bundle JAR. Note that this directory is different from the directory that stores the component configuration file for the local in-memory Oracle CEP caching provider (`META-INF/wlevs`).

- `tangosol-coherence-override.xml` is a global per-server file (referred to as "operational configuration" in the Oracle Coherence documentation); put this file in the Oracle CEP server `config` directory.

Update your application configuration file as Example 12–5 shows:

Example 12–5   Application Configuration File: Coherence Cache

```
<coherence-caching-system>
    <name>caching-system-id</name>
    <coherence-cache-config>
        ../wlevs/coherence/coherence-cache-config.xml
    </coherence-cache-config>
</coherence-caching-system>
```

When you declare that a caching system uses the Oracle Coherence provider, be sure that all of the caches of this caching system also map to an Oracle Coherence configuration and not an Oracle CEP local configuration, or Oracle CEP throws an exception.

12.3.1.1 The coherence-cache-config.xml File

The `coherence-cache-config.xml` file is the basic Oracle Coherence configuration file and must conform to the Oracle Coherence DTDs, as is true for any Oracle Coherence application.

The following sample shows a simple configuration. See the explanation after the sample for information about the sections in bold.

```
<?xml version="1.0"?>
<!DOCTYPE cache-config SYSTEM "cache-config.dtd">
<cache-config>
  <caching-scheme-mapping>
    <cache-mapping>
      <cache-name>myCoherenceCache</cache-name>
      <scheme-name>new-replicated</scheme-name>
    </cache-mapping>
    <cache-mapping>
      <cache-name>myLoaderCache</cache-name>
      <scheme-name>test-loader-scheme</scheme-name>
    </cache-mapping>
    <cache-mapping>
      <cache-name>myStoreCache</cache-name>
      <scheme-name>test-store-scheme</scheme-name>
    </cache-mapping>
  </caching-scheme-mapping>
  <caching-schemes>
    <replicated-scheme>
      <scheme-name>new-replicated</scheme-name>
      <service-name>ReplicatedCache</service-name>
      <backing-map-scheme>
        <local-scheme>
          <scheme-ref>my-local-scheme</scheme-ref>
        </local-scheme>
      </backing-map-scheme>
    </replicated-scheme>
```

```
        <local-scheme>
           <scheme-name>my-local-scheme</scheme-name>
           <eviction-policy>LRU</eviction-policy>
           <high-units>100</high-units>
           <low-units>50</low-units>
        </local-scheme>
        <local-scheme>
           <scheme-name>test-loader-scheme</scheme-name>
           <eviction-policy>LRU</eviction-policy>
           <high-units>100</high-units>
           <low-units>50</low-units>
           <cachestore-scheme>
              <class-scheme>
                 <class-factory-name>
                     com.bea.wlevs.cache.coherence.configuration.SpringFactory
                 </class-factory-name>
              <method-name>getLoader</method-name>
              <init-params>
                 <init-param>
                    <param-type>java.lang.String</param-type>
                    <param-value>{cache-name}</param-value>
                 </init-param>
              </init-params>
              </class-scheme>
           </cachestore-scheme>
        </local-scheme>
        <local-scheme>
           <scheme-name>test-store-scheme</scheme-name>
           <eviction-policy>LRU</eviction-policy>
           <high-units>100</high-units>
           <low-units>50</low-units>
           <cachestore-scheme>
             <class-scheme>
                 <class-factory-name>
                     com.bea.wlevs.cache.coherence.configuration.SpringFactory
                 </class-factory-name>
              <method-name>getStore</method-name>
              <init-params>
                 <init-param>
                     <param-type>java.lang.String</param-type>
                     <param-value>{cache-name}</param-value>
                 </init-param>
              </init-params>
             </class-scheme>
           </cachestore-scheme>
        </local-scheme>
     </caching-schemes>
</cache-config>
```

In the Oracle Coherence configuration file, the `cache-name` element that is a child element of `cache-mapping` identifies the name of the Oracle Coherence cache. The value of this element must exactly match the value of the `id` attribute of the `wlevs:cache` element in the EPN assembly file. For example, the following EPN assembly file snippet refers to the `myCoherenceCache` cache in the Oracle Coherence configuration file:

```
<wlevs:cache id="myCoherenceCache" advertise="false">
    <wlevs:caching-system ref="coherence-cache"/>
    <wlevs:cache-loader ref="localLoader"/>
    <wlevs:cache-listener ref="localListener"/>
</wlevs:cache>
```

The Oracle Coherence configuration file illustrates another requirement when using Oracle Coherence with Oracle CEP: an Oracle Coherence factory must be declared when using Spring to configure a loader or store for a cache. You do this using the

`cachestore-scheme` element in the Oracle Coherence configuration file to specify a factory class that allows Oracle Coherence to call into Oracle CEP and retrieve a reference to the loader or store that is configured for the cache. The only difference between configuring a loader or store is that the `method-name` element has a value of `getLoader` when a loader is used and `getStore` when a store is being used. You pass the cache name to the factory as an input parameter.

When declaring Oracle Coherence caches in the EPN assembly files of one or more applications deployed to the same Oracle CEP server, you should never configure multiple instances of the same cache with a loader or store. You might inadvertently do this by employing multiple applications that each configure the same Oracle Coherence cache with a loader or store in their respective EPN assembly file. If you do this, Oracle CEP throws an exception.

If multiple application bundles need to share Oracle Coherence caches, then you should put the EPN assembly file that contains the appropriate `wlevs:cache` and `wlevs:caching-system` in a separate bundle and set their `advertise` attributes to `true`.

Refer to your Oracle Coherence documentation (see http://www.oracle.com/technology/products/coherence/index.html) for detailed information about the `coherence-cache-config.xml` file.

12.3.1.2 The tangosol-coherence-override.xml File

The `tangosol-coherence-override.xml` file configures Oracle Coherence caching. You may includes this file if you are using Oracle Coherence for caching only. Do not include this file if you are using Oracle Coherence for clustering.

The following sample shows a simple configuration. See the explanation after the sample for information about the sections in bold.

```
<?xml version='1.0'?>
<coherence xml-override="/tangosol-coherence-override.xml">
  <cluster-config>
    <member-identity>
      <cluster-name>com.bea.wlevs.example.provider</cluster-name>
    </member-identity>
...
</coherence>
```

This configuration file is fairly standard. The main thing to note is that you should specify a `cluster-name` element to prevent Oracle Coherence from attempting to join existing Oracle Coherence clusters when Oracle CEP starts up; this can cause problems and sometimes even prevent Oracle CEP from starting.

Refer to your Oracle Coherence documentation (see http://www.oracle.com/technology/products/coherence/index.html) for detailed information about the `tangosol-coherence-override.xml` file.

For more information on Oracle CEP clusters, see "Administrating Oracle CEP Multi-Server Domains" in the *Oracle CEP Administrator's Guide*.

12.3.2 Configuring an Oracle Coherence Cache as an Event Listener

An Oracle Coherence cache can be configured as an explicit listener in the event processing network in order to receive events.

For example, to specify that a cache listens to a channel, specify the `wlevs:listener` element with a reference to the cache as a child of the `wlevs:channel` element as shown below:

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

As the channel sends new events to the cache, they are inserted into the cache. If a *remove event* (an old event that exits the output window) is sent by the channel, then the event is removed from the cache.

12.3.2.1 Specifying the Key Used to Index an Oracle Coherence Cache

When you configure an Oracle Coherence cache to be a listener, events are inserted into the cache. This section describes the variety of options available to you to specify the key used to index a cache in this instance.

If you do not explicitly specify a key, the event object itself serves as both the key and value when the event is inserted into the cache. In this case, the event class must include a valid implementation of the `equals` and `hashcode` methods that take into account the values of the key properties.

See the following for ways to explicitly specify a key:

- Section 12.3.2.1.1, "Specifying a Key Property in EPN Assembly File"
- Section 12.3.2.1.2, "Using a Metadata Annotation to Specify a Key"
- Section 12.3.2.1.3, "Specifying a Composite Key"

12.3.2.1.1 Specifying a Key Property in EPN Assembly File The first option is to specify a property name for the key property when a cache is declared in the EPN assembly file using the `key-properties` attribute, as shown in the following example:

```
<wlevs:cache id="myCache" key-properties="key-property-name">
    <wlevs:caching-system ref="caching-system-id"/>
</wlevs:cache>
```

In this case, all events that are inserted into the cache are required to have a property of this name at runtime, otherwise Oracle CEP throws an exception.

For example, assume the event type being inserted into the cache looks something like the following; note the `key` property (only relevant Java source shown):

```
public class MyEvent {
    private String key;
    public MyEvent() {
    }
    public MyEvent(String key) {
        this.key = key;
    }
    public String getKey() {
        return key;
    }
    public void setKey(String key) {
        this.key = key;
    }
...
}
```

The corresponding declaration in the EPN assembly file would look like the following:

```
<wlevs:cache id="myCache" key-properties="key">
    <wlevs:caching-system ref="caching-system-id"/>
</wlevs:cache>
```

12.3.2.1.2 Using a Metadata Annotation to Specify a Key The second option is to use the metadata annotation `com.bea.wlevs.ede.api.Key` to annotate the event property in the Java class that implements the event type. This annotation does not have any attributes.

The following example shows how to specify that the `key` property of the `MyEvent` event type is the key; only relevant code is shown:

```
import com.bea.wlevs.ede.api.Key;
public class MyEvent {
    @Key
    private String key;
    public MyEvent() {
    }
    public MyEvent(String key) {
        this.key = key;
    }
    public String getKey() {
        return key;
    }
    public void setKey(String key) {
        this.key = key;
    }
 ...
}
```

12.3.2.1.3 Specifying a Composite Key The final option is to use the `key-class` attribute of the `<wlevs:cache>` element to specify a composite key in which multiple properties form the key. The value of the `key-class` attribute must be a JavaBean whose public fields match the fields of the event class. The matching is done according to the field name. For example:

```
<wlevs:cache id="myCache" key-class="key-class-name">
    <wlevs:caching-system ref="caching-system-id"/>
</wlevs:cache>
```

12.3.3 Configuring an Oracle Coherence Cache as an Event Source

You can configure an Oracle Coherence cache as a source of events to which another component in the event processing network listens. The listening component can be an adapter or a standard Spring bean. Any component that listens to a cache must implement the `com.tangosol.util.MapListener` interface. The following example shows how to configure an Oracle Coherence cache to be an event source for a Spring bean:

```
<wlevs:coherence-caching-system id="caching-system-id"/>
...
<wlevs:cache id="myCache" advertise="false">
    <wlevs:caching-system ref="caching-system-id"/>
    <wlevs:cache-loader ref="localLoader"/>
    <wlevs:cache-listener ref="localListener"/>
</wlevs:cache>
<bean id="localListener"
      class="com.bea.wlevs.example.provider.coherence.LocalListener"/>
```

In the example, the `cache-listener-id` Spring bean listens to events coming from the cache; the class that implements this component, `com.bea.wlevs.example.provider.coherence.LocalListener`, must implement the appropriate Oracle Coherence-specific Java interfaces such as

com.tangosol.util.MapListener. You must program this LocalListener class yourself as Example 12–6 shows.

Example 12–6   Oracle Coherence Cache LocalListener Implementation

```
package com.bea.wlevs.example.provider.coherence;

import com.tangosol.util.MapEvent;
import com.tangosol.util.MapListener;

public class LocalListener implements MapListener {
    public static int deleted = 0;
    public static int inserted = 0;
    public static int updated = 0;

    public void entryDeleted(MapEvent event) {
        deleted++;
    }
    public void entryInserted(MapEvent event) {
        inserted++;
    }
    public void entryUpdated(MapEvent event) {
        updated++;
    }
}
```

12.3.4 Configuring an Oracle Coherence Cache Loader or Store

Using an Oracle Coherence cache, you may configure either a wlevs:cache-loader or a wlevs:cache-store child element of the wlevs:cache element in the EPN assembly file, but not both. This is because Oracle Coherence combines the loader and store into a single component:

- Specify a cache loader when the backing store is read-only.

  See Section 12.3.4.1, "Configuring an Oracle Coherence Cache Loader."

- Specify a cache store when the backing store is read-write.

  See Section 12.3.4.2, "Configuring an Oracle Coherence Cache Store."

12.3.4.1 Configuring an Oracle Coherence Cache Loader

In Example 12–7, the localLoader Spring bean loads events into the oracle Coherence cache when the backing store is read-only. If the backing store is read-write, use a cache store (see Section 12.3.4.2, "Configuring an Oracle Coherence Cache Store").

The class that implements this component, com.bea.wlevs.example.provider.coherence.LocalLoader, must implement the appropriate Oracle Coherence-specific Java interfaces such as com.tangosol.net.cache.CacheLoader. You must program this LocalLoader class yourself as Example 12–8 shows.

Example 12–7   Oracle Coherence Cache EPN Assembly File for a Cache Loader

```
<wlevs:coherence-caching-system id="caching-system-id"/>
<wlevs:cache id="myCache" advertise="false">
    <wlevs:caching-system ref="caching-system-id"/>
    <wlevs:cache-loader ref="localLoader"/>
</wlevs:cache>
<bean id="localLoader"
      class="com.bea.wlevs.example.provider.coherence.LocalLoader"/>
```

Example 12–8 Oracle Coherence Cache LocalLoader Implementation

```
package com.bea.wlevs.example.provider.coherence;

import java.util.Collection;
import java.util.HashMap;
import java.util.HashSet;
import java.util.Map;
import java.util.Set;

import com.bea.wlevs.example.provider.event.ProviderData;
import com.tangosol.net.cache.CacheLoader;

public class LocalLoader implements CacheLoader {
    public static int loadCount = 0;
    public static Set keys = new HashSet();

    public LocalLoader() {
    }
    public Object load(Object key) {
        loadCount++;
        keys.add(key);
        return new ProviderData((String) key);
    }
    public Map loadAll(Collection keys) {
        Map result = new HashMap();

        for (Object key : keys) {
            result.put(key, load(key));
        }
        return result;
    }
}
```

12.3.4.2 Configuring an Oracle Coherence Cache Store

In Example 12–9, the `localStore` Spring bean loads events into the cache when the backing store is read-write. If the backing store is read-only, use a cache loader (see Section 12.3.4.1, "Configuring an Oracle Coherence Cache Loader").

The class that implements this component, `com.bea.wlevs.example.provider.coherence.LocalStore`, must implement the appropriate Oracle Coherence-specific Java interfaces such as `com.tangosol.net.cache.CacheStore`. You must program this `LocalStore` class yourself as Example 12–10 shows.

Example 12–9 Oracle Coherence Cache EPN Assembly File for a Cache Store

```
<wlevs:coherence-caching-system id="caching-system-id"/>
<wlevs:cache id="myCache" advertise="false">
    <wlevs:caching-system ref="caching-system-id"/>
    <wlevs:cache-store ref="localStore"/>
</wlevs:cache>
<bean id="localStore"
      class="com.bea.wlevs.example.provider.coherence.LocalStore"/>
```

Example 12–10 Oracle Coherence Cache LocalStore Implementation

```
package com.bea.wlevs.example.provider.coherence;

import java.util.Collection;
import java.util.HashMap;
import java.util.Map;
```

```
import java.util.Set;

import com.bea.wlevs.example.provider.event.ProviderData;
import com.tangosol.net.cache.CacheStore;

public class LocalStore implements CacheStore {
    public static int eraseCount = 0;
    public static int storeCount = 0;
    public static int loadCount = 0;

    public void erase(Object key) {
        eraseCount++;
    }
    public void eraseAll(Collection keys) {
        for (Object key : keys) {
            erase(key);
        }
    }
    public void store(Object key, Object value) {
        //
        // Do the store operation here.
        //
    }
    public void storeAll(Map entries) {
        for (Map.Entry entry : (Set <Map.Entry>)entries.entrySet()) {
            store(entry.getKey(), entry.getValue());
        }
    }
    public Object load(Object key) {
        loadCount++;
        return new ProviderData((String) key);
    }
    public Map loadAll(Collection keys) {
        Map result = new HashMap();
        for (Object key : keys) {
            result.put(key, load(key));
        }
        return result;
    }
}
```

12.4 Configuring a Third-Party Caching System and Cache

You can configure your application to use a third-party caching system and cache.

> **Note:** It is assumed in this section that you have already created an
> Oracle CEP application along with its EPN assembly file and that you
> want to update the application to use caching. If you have not, refer to
> Chapter 1, "Overview of Creating Oracle CEP Applications" for
> details.

To configure a third-party caching system and cache:

1. Create a plug-in to define the third-party caching system as an Oracle CEP caching
    system provider.

    This involves:

    - Implementing the `com.bea.wlevs.cache.spi.CachingSystem` interface

    - Creating a factory that creates caching systems of this type.

    - Registering the factory with an attribute that identifies its provider type.

2. Declare the caching system in the EPN assembly file.

Use the `wlevs:caching-system` element to declare a third-party implementation; use the `class` or `provider` attributes to specify additional information.

For simplicity, you can include the third-party implementation code inside the Oracle CEP application bundle itself to avoid having to import or export packages and managing the lifecycle of a separate bundle that contains the third-party implementation. In this case the `wlevs:caching-system` element appears in the EPN assembly file as shown in the following example:

```
<wlevs:caching-system id="caching-system-id"
                      class="third-party-implementation-class"/>
```

The `class` attribute specifies a Java class that must implement the `com.bea.wlevs.cache.spi.CachingSystem` interface. For details about this interface, see the *Oracle CEP Java API Reference*.

Sometimes, however, you might not be able, or want, to include the third-party caching implementation in the same bundle as the Oracle CEP application that is using it. In this case, you must create a *separate* bundle whose Spring application context includes the `wlevs:caching-system` element, with the `advertise` attribute mandatory:

```
<wlevs:caching-system id ="caching-system-id"
                      class="third-party-implementation-class"
                      advertise="true"/>
```

Alternatively, if you want to decouple the implementation bundle from the bundle that references it, or you are plugging in a caching implementation that supports multiple caching systems per Java process, you can specify 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>
```

The factory class (`the.factory.class.name` in the example) must implement the `com.bea.wlevs.cache.spi.CachingSystemFactory` interface. This interface has a single method, `create()`, that returns a `com.bea.wlevs.cache.spi.CachingSystem` instance.

You must deploy this bundle alongside the application bundle so that the latter can start using it.

3. Declare one or more caches for this caching system in the EPN assembly file.

After you have declared a caching system for an application, you configure one or more caches using the `wlevs:cache` element:

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

The `name` attribute is optional; specify it only if the name of the cache in the caching system is different from its ID. The `wlevs:caching-system` child element references the already-declared caching system that contains the cache. You must specify this child element only if the caching system is ambiguous: there

is more than one caching system declared (either implicitly or explicitly) or if the caching system is in a different application or bundle.

You can export both the caching system and the cache as an OSGI service using the `advertise` attribute.

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

If the cache is advertised, then a component in the EPN of an application in a *separate* bundle can then reference it. The following example shows how a processor in one bundle can use as a cache source the cache with ID `cache-id` located in a separate bundle (called `cacheprovider`):

```
<wlevs:processor id="myProcessor2">
    <wlevs:cache-source ref="cacheprovider:cache-id"/>
</wlevs:processor>
```

The caching system is responsible for creating the cache associated with a particular name and returning a reference to the cache. The resulting cache bean implements the `java.util.Map` interface.

4. Configure the third-party caching system and its caches by updating the third-party caching configuration file or files for the application.

Refer to your third-party cache documentation.

5. Optionally, override the default third-party cache configuration by updating the appropriate configuration file with one or more additional `cache` element child elements.

■ Specify that a cache is an event sink by configuring it as a listener to another component in the event processing network.

Refer to your third-party cache documentation.

■ Specify that a cache is an event source to which another component in the event processing network listens.

Refer to your third-party cache documentation.

■ Configure a cache loader or store.

Refer to your third-party cache documentation.

6. Access the third-party cache:

– Optionally reference the third-party cache in a query statement.

See:

* Section 12.5, "Accessing a Cache From an Oracle CQL Statement"

* Section 12.6, "Accessing a Cache From an EPL Statement"

– Optionally configure and program a custom adapter, business POJO, or Oracle CQL or EPL user-defined function to access the third-party cache.

The configuration is done in the EPN assembly file and the programming is done in the Java file that implements the adapter, POJO, or user-defined function.

See:

         * Section 12.7, "Accessing a Cache From an Adapter"

         * Section 12.8, "Accessing a Cache From a Business POJO"

         * Section 12.9, "Accessing a Cache From an Oracle CQL User-Defined Function"

         * Section 12.10, "Accessing a Cache From an EPL User-Defined Function"

    – Optionally, access the third-party cache using JMX.

    See Section 12.11, "Accessing a Cache Using JMX".

7. When you assemble your application, verify that the `META-INF/MANIFEST.MF` file includes the following import as Figure 12–1 shows:

```
com.bea.wlevs.cache.spi; version ="11.1.0.0"
```

If the `MANIFEST.MF` files does not include this import, update the `MANIFEST.MF` file to add this import before deploying your application.

Figure 12–3   Editing the MANIFEST.MF File
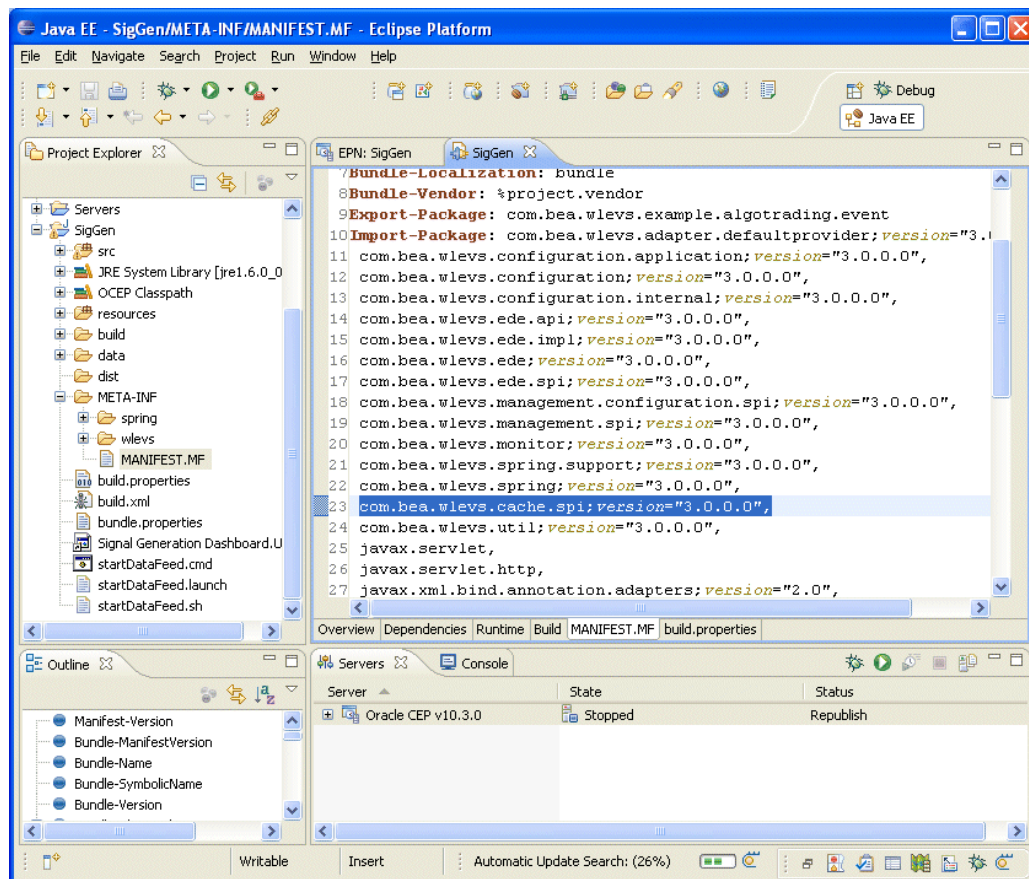
12.5 Accessing a Cache From an Oracle CQL Statement

You can reference a cache from an Oracle CQL statement in much the same way you reference a channel; this feature enables you to enrich standard streaming data with data from a separate source. Example 12–11 shows a valid Oracle CQL query that joins trade events from a standard channel named `S1` with stock symbol data from a cache named `stockCache`:

Example 12–11   Valid Oracle CQL Query Against a Cache

```
SELECT  S1.symbol, S1.lastPrice, stockCache.description
FROM    S1 [Now], stockCache
WHERE   S1.symbol = stockCache.symbol
```

Whenever you query a cache, you must join against the `[Now]` window. This guarantees that the query will execute against a snapshot of the cache. If you join against any other window type, then if the cache changes before the window expires, the query will be incorrect.

Example 12–12 shows an invalid Oracle CQL query that joins a `Range` window against a cache. If the cache changes before this window expires, the query will be incorrect. Consequently, this query will raise Oracle CEP server error "`external relation must be joined with s[now]`".

Example 12–12   Invalid Oracle CQL Query Against a Cache

```
SELECT trade.symbol, trade.price, trade.numberOfShares, company.name
FROM TradeStream [Range 8 hours] as trade, CompanyCache as company
WHERE trade.symbol = company.id
```

When you use data from a cache in an Oracle CQL query, Oracle CEP *pulls* the data rather than it being *pushed*, as is the case with a channel. This means that, continuing with Example 12–11, the query executes only when a channel pushes a trade event to the query; the stock symbol data in the cache never causes a query to execute, it is only pulled by the query when needed.

You must abide by these restrictions when using a cache in an Oracle CQL query:

- You must specify the key properties for data in the cache.

    For instructions on specifying the cache key, see:

    – Section 12.2.1.1, "Specifying the Key Used to Index an Oracle CEP Local Cache"

    – Section 12.3.2.1, "Specifying the Key Used to Index an Oracle Coherence Cache"

- Joins must be executed only by referencing the cache key.

- You cannot use a cache in a view. Instead, use a join.

- Only a single channel source may occur in the `FROM` clause of an Oracle CQL statement that joins cache data source(s).

12.5.1 How to Access a Cache From an Oracle CQL Statement

This section describes how to reference a cache in an Oracle CQL query statement.

This procedure assumes that you have already configured the caching system and caches. For more information, see:

- Section 12.2, "Configuring an Oracle CEP Local Caching System and Cache"

- Section 12.3, "Configuring an Oracle Coherence Caching System and Cache"

- Section 12.4, "Configuring a Third-Party Caching System and Cache"

To access a cache from an Oracle CQL statement:

1. If you have not already done so, create the event type that corresponds to the cache data and register it in the event repository.

See Section 1.5, "Creating Oracle CEP Event Types."

2. Specify the key properties for the data in the cache.

    There are a variety of ways to do this.

    For instructions on specifying the cache key, see:

    –   Section 12.2.1.1, "Specifying the Key Used to Index an Oracle CEP Local Cache"

    –   Section 12.3.2.1, "Specifying the Key Used to Index an Oracle Coherence Cache"

3. In the EPN assembly file, update the configuration of the cache to declare the event type of its values; use the `value-type` attribute of the `wlevs:cache` element. For example:

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

    The value-type attribute specifies the type for the values contained in the cache. This must be a valid type name in the event type repository.

    This attribute is required only if the cache is referenced in an Oracle CQL query. This is because the query processor needs to know the type of events in the cache.

4. In the EPN assembly file, update the configuration of the processor that executes the Oracle CQL query that references a cache, adding a `wlevs:cache-source` child element that references the cache. For example:

```
<wlevs:channel 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` channel as usual; however, the Oracle CQL 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 `CompanyEvent`, the processor pulls instances of that event type from the cache.

12.6 Accessing a Cache From an EPL Statement

You can reference a cache from an EPL statement in much the same way you reference a channel; this feature enables you to enrich standard streaming data with data from a separate source. For example, the following EPL query joins trade events from a standard channel with company data from a cache:

```
INSERT INTO EnrichedTradeEvent
SELECT trade.symbol, trade.price, trade.numberOfShares, company.name
FROM TradeEvent trade RETAIN 8 hours, Company company
WHERE trade.symbol = company.id
```

In the example, both `TradeEvent` and `Company` are event types registered in the repository, but they have been configured in such a way that `TradeEvents` come from a standard stream of events but `Company` maps to a cache in the event processing network. This configuration happens outside of the EPL query, which means that the source of the data is transparent in the query itself.

When you use data from a cache in an EPL query, Oracle CEP *pulls* the data rather than it being *pushed*, as is the case with a channel. This means that, continuing with the preceding sample, the query executes only when a channel pushes a trade event to the query; the company data in the cache never causes a query to execute, it is only pulled by the query when needed.

You must abide by these restrictions when using a cache in an EPL query:

- You must specify the key properties for data in the cache.

  For instructions on specifying the cache key, see:

  - Section 12.2.1.1, "Specifying the Key Used to Index an Oracle CEP Local Cache"

  - Section 12.3.2.1, "Specifying the Key Used to Index an Oracle Coherence Cache"

- Joins must be executed only by referencing the cache key.

- You cannot specify a `RETAIN` clause for data pulled from a cache. If an event type that gets its data from a cache is included in a `RETAIN` clause, Oracle CEP ignores it.

- You cannot use a cache in a correlated sub-query. Instead, use a join.

- Only a single channel source may occur in the `FROM` clause of an EPL statement that joins cache data source(s). Using multiple cache sources and parameterized SQL queries is supported.

12.6.1 How To Access a Cache From an EPL Statement

This section describes how to reference a cache in an EPL query statement.

This procedure assumes that you have already configured the caching system and caches. For more information, see:

- Section 12.2, "Configuring an Oracle CEP Local Caching System and Cache"

- Section 12.3, "Configuring an Oracle Coherence Caching System and Cache"

- Section 12.4, "Configuring a Third-Party Caching System and Cache"

To access a cache from an EPL statement:

1. If you have not already done so, create the event type that corresponds to the cache data, such as `Company` in the preceding example, and registered it in the event repository. See Section 1.5, "Creating Oracle CEP Event Types."

2. Specify the key properties for the data in the cache. There are a variety of ways to do this; see

   - Section 12.2.1.1, "Specifying the Key Used to Index an Oracle CEP Local Cache"

   - Section 12.3.2.1, "Specifying the Key Used to Index an Oracle Coherence Cache"

3. In the EPN assembly file, update the configuration of the cache in the EPN assembly file to declare the event type of its values; use the `value-type` attribute of the `wlevs:cache` element. For example:

```
<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>
```

The value-type attribute specifies the type for the values contained in the cache. This must 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.

4. In the EPN assembly file, update the configuration of the processor that executes the EPL query that references a cache, adding a `wlevs:cache-source` child element that references the cache. For example:

```
<wlevs:channel 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` channel 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.

12.7 Accessing a Cache From an Adapter

An adapter can also be injected with a cache using the standard Spring mechanism for referencing another bean. A cache bean implements the `java.util.Map` interface which is what the adapter uses to access the injected cache.

First, the configuration of the adapter in the EPN assembly file must be updated with a `wlevs:instance-property` child element, as shown in the following example:

```
<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>
...
<wlevs:adapter id="myAdapter" provider="myProvider">
    <wlevs:instance-property name="map" ref="cache-id"/>
</wlevs:adapter>
```

In the example, the `ref` attribute of `wlevs:instance-property` references the `id` value of the `wlevs:cache` element. Oracle CEP automatically injects the cache, implemented as a `java.util.Map`, into the adapter.

In the adapter Java source, add a `setMap (Map)` method with the code that implements whatever you want the adapter to do with the cache:

```
package com.bea.wlevs.example;
…
```

```
import java.util.Map;
public class MyAdapter implements Runnable, Adapter, EventSource, SuspendableBean  {
...
    public void setMap (Map map) {...}
}
```

12.8 Accessing a Cache From a Business POJO

A business POJO, configured as a standard Spring bean in the EPN assembly file, can be injected with a cache using the standard Spring mechanism for referencing another bean. In this way the POJO can view and manipulate the cache. A cache bean implements the `java.util.Map` interface which is what the business POJO uses to access the injected cache. A cache bean can also implement a vendor-specific sub-interface of `java.util.Map`, but for portability it is recommended that you implement `Map`.

First, the configuration of the business POJO in the EPN assembly file must be updated with a `property` child element, as shown in the following example based on the Output bean of the FX example (see "HelloWorld Example" in the *Oracle CEP Getting Started*):

```
<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>
...
<bean class="com.bea.wlevs.example.helloworld.HelloWorldBean">
    <property name="map" ref="cache-id"/>
</bean>
```

In the example, the `ref` attribute of the `property` element references the `id` value of the `wlevs:cache` element. Oracle CEP automatically injects the cache, implemented as a `java.util.Map`, into the business POJO bean.

In the business POJO bean Java source, add a `setMap (Map)` method with the code that implements whatever you want the POJO to do with the cache:

```
package com.bea.wlevs.example.helloworld;
…
import java.util.Map;
public class HelloWorldBean implements EventSink {
...
    public void setMap (Map map) {...}
}
```

12.9 Accessing a Cache From an Oracle CQL User-Defined Function

In addition to standard event streams, Oracle CQL rules can also invoke the member methods of a user-defined function.

These user-defined functions are implemented as standard Java classes and are declared in the component configuration file of the Oracle CQL processor, as shown in the following example:

```
<bean id="orderFunction" class="orderFunction-impl-class"/>
```

The processor in which the relevant Oracle CQL rule runs must then be injected with the user-defined function using the `wlevs:function` child element, referencing the Spring with the `ref` attribute:

```
<wlevs:processor id= "tradeProcessor">
```

```
    <wlevs:function ref="orderFunction"/>
</wlevs:processor>
```

The following Oracle CQL rule, assumed to be configured for the tradeProcessor processor, shows how to invoke the existsOrder() method of the orderFunction user-defined function:

```
INSERT INTO InstitutionalOrder
SELECT er.orderKey AS key, er.symbol AS symbol, er.shares as cumulativeShares
FROM ExecutionRequest er [Range 8 hours]
WHERE NOT orderFunction.existsOrder(er.orderKey)
```

You can also configure the user-defined function to access a cache by injecting the function with a cache using the standard Spring mechanism for referencing another bean. A cache bean implements the java.util.Map interface which is what the user-defined function uses to access the injected cache.

First, the configuration of the user-defined function in the EPN assembly file must be updated with a wlevs:property child element, as shown in the following example:

```
<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>
    ...
<bean id="orderFunction" class="orderFunction-impl-class">
    <wlevs:property name="cache" ref="cache-id"/>
</bean>
```

In the example, the ref attribute of the wlevs:property element references the id value of the wlevs:cache element. Oracle CEP automatically injects the cache, implemented as a java.util.Map, into the user-defined function.

In the user-defined function's Java source, add a setMap (Map) method with the code that implements whatever you want the function to do with the cache:

```
package com.bea.wlevs.example;
…
import java.util.Map;
public class OrderFunction {
...
    public void setMap (Map map) {...}
}
```

12.10 Accessing a Cache From an EPL User-Defined Function

In addition to standard event streams, EPL rules can also invoke the member methods of a user-defined function.

These user-defined functions are implemented as standard Java classes and are declared in the EPN assembly file using the standard Spring bean tags, as shown in the following example:

```
<bean id="orderFunction" class="orderFunction-impl-class"/>
```

The processor in which the relevant EPL rule runs must then be injected with the user-defined function using the wlevs:function child element, referencing the Spring with the ref attribute:

```
<wlevs:processor id= "tradeProcessor">
    <wlevs:function ref="orderFunction"/>
</wlevs:processor>
```

The following EPL rule, assumed to be configured for the `tradeProcessor` processor, shows how to invoke the `existsOrder()` method of the `orderFunction` user-defined function:

```
INSERT INTO InstitutionalOrder
SELECT er.orderKey AS key, er.symbol AS symbol, er.shares as cumulativeShares
FROM ExecutionRequest er RETAIN 8 HOURS WITH UNIQUE KEY
WHERE NOT orderFunction.existsOrder(er.orderKey)
```

You can also configure the user-defined function to access a cache by injecting the function with a cache using the standard Spring mechanism for referencing another bean. A cache bean implements the `java.util.Map` interface which is what the user-defined function uses to access the injected cache.

First, the configuration of the user-defined function in the EPN assembly file must be updated with a `wlevs:property` child element, as shown in the following example:

```
<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>
...
<bean id="orderFunction" class="orderFunction-impl-class">
    <wlevs:property name="cache" ref="cache-id"/>
</bean>
```

In the example, the `ref` attribute of the `wlevs:property` element references the `id` value of the `wlevs:cache` element. Oracle CEP automatically injects the cache, implemented as a `java.util.Map`, into the user-defined function.

In the user-defined function's Java source, add a `setMap (Map)` method with the code that implements whatever you want the function to do with the cache:

```
package com.bea.wlevs.example;
…
import java.util.Map;
public class OrderFunction {
...
    public void setMap (Map map) {...}
}
```

12.11 Accessing a Cache Using JMX

At runtime, you can access a cache programatically using JMX and the MBeans that Oracle CEP deploys for the caching systems and caches you define.

This section describes:

- Section 12.11.1, "How to Access a Cache With JMX Using Oracle CEP Visualizer"
- Section 12.11.2, "How to Access a Cache With JMX Using Java"

For more information, "Configuring JMX for Oracle CEP" in the *Oracle CEP Administrator's Guide*

12.11.1 How to Access a Cache With JMX Using Oracle CEP Visualizer

The simplest and least error-prone way to access a caching system or cache with JMX is to use the Oracle CEP Visualizer.

For more information, see "Server and Domain Tasks" in the *Oracle CEP Visualizer User's Guide*.

12.11.2 How to Access a Cache With JMX Using Java

The simplest and least error-prone way to access a caching system or cache with JMX is to use the Oracle CEP Visualizer (see Section 12.11.1, "How to Access a Cache With JMX Using Oracle CEP Visualizer"). Alternatively, you can access a caching system or cache with JMX using Java code that you write.

Oracle CEP creates a `StageMBean` for each cache that your application uses as a stage. The `Type` of this MBean is `Stage`.

To access a cache with JMX using Java:

1. Connect to the JMX service that Oracle CEP server provides.

    For more information, see "Configuring JMX for Oracle CEP" in the *Oracle CEP Administrator's Guide*

2. Get a list of cache `StageMbean` using either of:

    -   `CachingSystemMBean.getCacheMBeans()`

    -   `ApplicationMBean.getStageMBeans()`

3. Get the `ObjectName` for a given `StageMBean` that represents a cache in your caching system:

    ```
    ObjectName cacheName = ObjectName.getInstance ('com.bea.wlevs:Name =
    newCache,Type=Stage,CachingSystem=newCachingSystem,Application=provider');
    ```

4. Get a proxy instance for the `StageMBean` with this `ObjectName`:

    ```
    StageMBean cache = (StageMBean) MBeanServerInvocationHandler.newProxyInstance(
        server, cacheName, StageMBean.class, false
    );
    ```

5. Use the methods of the `StageMBean` to access the cache.

13

Configuring Event Record and Playback

This section contains information on the following subjects:

- Section 13.1, "Overview of Configuring Event Record and Playback"
- Section 13.2, "Configuring Event Record and Playback in Your Application"
- Section 13.3, "Creating a Custom Event Store Provider"

13.1 Overview of Configuring Event Record and Playback

Oracle CEP event repository feature allows you to persist the events that flow out of a component of the event processing network (EPN) to a store, such as a database table, and then play them back at a later stage or explicitly query the events from a component such as an event bean.

A typical use case of this feature is the ability to debug a problem with a currently running application. If you have been recording the events at a node in the EPN when the problem occurred, you can later playback the same list of events to recreate the problem scenario for debugging purposes.

The following graphic shows the EPN of the Event Record and Playback example and demonstrates at what point events are recorded and where they are played back. The `simpleEventSource` adapter has been configured to record events; as indicated, the record happens as events flow *out* of the adapter. The `eventStream` channel has been configured to playback events; as indicated, the playback happens at the point where events flow *into* the channel.

Figure 13–1   Configuring Record and Playback in an EPN
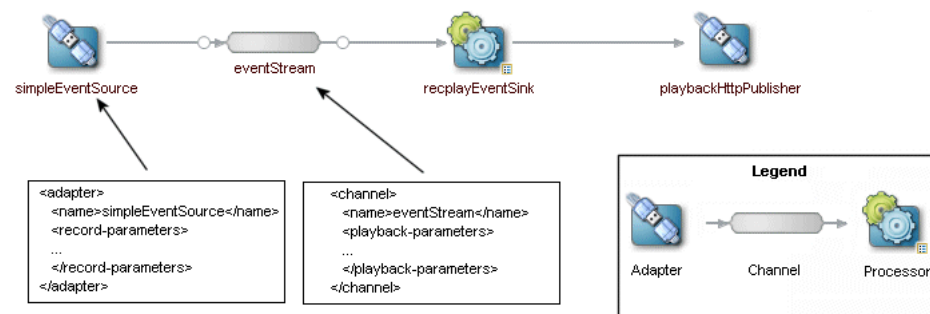
13.1.1 Storing Events in the Persistent Event Store

Oracle CEP provides an RDBMS-based implementation of the event store; this implementation stores data in a relational database. You can use either the database

server included with Oracle CEP (Apache Derby) or another database server, such as Oracle DBMS.

Oracle CEP includes a default database server, Apache Derby (see http://db.apache.org/derby/index.html), that you can use to record events. Apache Derby is an open source relational database implemented entirely in Java. By default, Derby creates its database files and log file (derby.log) from the directory in which you started Oracle CEP, such as *DOMAIN_DIR/servername*. You can change this default location by setting the system property derby.system.home to a different directory.

You can also create a custom event store provider to store events in a non-RDBMS persistent store. For details, see Section 13.3, "Creating a Custom Event Store Provider."

13.1.2 Recording Events

You can configure recording for any component in the event processing network (EPN) that produces events: processors, adapters, streams, and event beans. Processors and streams always produce events; adapters and event beans must implement the EventSource interface. Additionally, you can configure that events from different components in the EPN be stored in different persistent stores, or that all events go to the same store. Note that only events that are outputted by the component are recorded.

You enable the recording of events for a component by updating its configuration file and adding the record-parameters element. Using the child elements of record-parameters, you specify the event store to which the events are recorded, an initial time period when recording should take place, the list of event types you want to store, and so on.

After you deploy the application and events start flowing through the network, recording begins either automatically because you configured it to start at a certain time or because you dynamically start it using administration tools. For each component you have configured for recording, Oracle CEP stores the events that flow out of it to the appropriate store along with a timestamp of the time it was recorded.

13.1.3 Playing Back Events

You can configure playback for any component in the event processing network (EPN): processors, adapters, streams, and event beans. Typically the playback component is a node later in the network than the node that recorded the events.

You enable the playback of events for a component by updating its configuration file and adding the playback-parameters element. Using the child elements of playback-parameters, you specify the event store from which the events are played back, the list event types you want to play back (by default all are played back), the time range of the recorded events you want to play back, and so on. By default, Oracle CEP plays back the events in a time accurate manner; however, you can also configure that the events get played back either faster or slower than they originally flowed out of the component from which they were recorded.

After you deploy the application and events start flowing through the network, you must start the playback by using the administration tools (Oracle CEP Visualizer or wlevs.Admin). Oracle CEP reads the events from the appropriate persistent store and inserts them into the appropriate place in the EPN.

It is important to note that when a component gets a playback event, it looks exactly like the original event. Additionally, a component later in the network has been

configured to record events, then Oracle CEP records the playback events as well as the "real" events.

For more information, see:

- "Recording and Playing Back Events Flowing Through an EPN" in the *Oracle CEP Visualizer User's Guide*

- "Commands for Controlling Event Record and Playback" in the *Oracle CEP Administrator's Guide*

13.1.4 Querying Stored Events

You can use the event store API to query a store for past events given a record time range and the component from which the events were recorded. The actual query you use depends on the event repository provider; for example, you would use Oracle CQL or EPL for the RDBMS provider included in Oracle CEP. You can also use these APIs to delete old events from the event store.

13.1.5 Restrictions on the Event Types that Can Be Recorded

In this release, the following restrictions apply to the event types that can be recorded to the event store if you use the Oracle RDBMS-based provider:

- The fields of the event type must be simple data types. In other words, the fields cannot be complex types such as arrays, `java.util.Map`, or nested objects.

- The simple fields must be one of the supported Java types, as listed in Table 13–3.

13.1.6 Record and Playback Example

The sample code in this section is taken from the event record and playback example, located in the *ORACLE_CEP_HOME*\ocep_ 11.1\samples\source\applications\recplay directory, where *ORACLE_CEP_ HOME* refers to the main Oracle CEP installation directory, such as d:\oracle_cep.

For details about running and building the example, see "Event Record and Playback Example" in the *Oracle CEP Getting Started*.

13.2 Configuring Event Record and Playback in Your Application

Depending on how you are going to use the event repository, there are different tasks that you must perform, as described in the following procedure that in turn point to sections with additional details.

> **Note:** It is assumed in this section that you have already created an Oracle CEP application along with component configuration file(s) for the components and that you want to update the application so that components record or playback events. If you have not, refer to Chapter 1, "Overview of Creating Oracle CEP Applications," for details.

To configure record and playback of events in your application:

1. Configure an event store for your Oracle CEP server instance.

You must perform this configuration step even if you plan to use the event store provider provided by Oracle CEP, Apache Derby (see http://db.apache.org/derby/index.html).

See Section 13.2.1, "Configuring an Event Store for Oracle CEP Server."

2. Configure a component in your EPN to record events by updating the component's configuration file.

   The component can be a processor, adapter, channel, or event bean. Only events flowing out of the component are recorded.

   See Section 13.2.2, "Configuring a Component to Record Events."

3. Configure a component in your EPN to playback events by updating the component's configuration file.

   The component can be a processor, adapter, channel, or event bean. Only components that are also event sinks can playback events; events are played to the input side of the component.

   See Section 13.2.3, "Configuring a Component to Playback Events."

4. Redeploy your application for the changes to take effect.

5. If you have not specified an explicit start and end time for recording events, you must use Oracle CEP Visualizer or `wlevs.Admin` to start recording. You must always use these administration tools to start and end the playback of events.

   See Section 13.2.4, "Starting and Stopping the Record and Playback of Events."

13.2.1 Configuring an Event Store for Oracle CEP Server

Oracle CEP stores recorded events in a database, which means that before you can start using the record and playback feature in your own application, you must specify where the database server is located along with the name of the database server that will contain the recorded events. You do this by updating the `config.xml` file of your Oracle CEP server instance, as described in later procedures.

You do not, however, create the actual tables that store the recorded events; Oracle CEP automatically does this for you after you deploy the application that uses the record and playback feature and recording starts to take place. See Section 13.2.5, "Description of the Database Tables Created by the RDMBS Provider" for information about how Oracle CEP creates these tables.

To configure an event store for Oracle CEP server:

1. Stop your Oracle CEP server instance, if it is running.

2. Using your favorite XML editor, open the server's `config.xml` file for edit.

   The `config.xml` file is located in the *DOMAIN_DIR*/*servername*/`config` directory of your server, where *DOMAIN_DIR* refers to the domain directory, such as `/oracle_cep/user_projects/domains/myDomain` and *servername* refers to the name of your server, such as `defaultserver`.

3. Configure access to your relational database by adding a `data-source` element to the `config.xml` file.

   For details, see "Configuring JDBC for Oracle CEP" in the *Oracle CEP Administrator's Guide*.

   If you want to use the default Apache Derby database, add the following `data-source` element:

```
<?xml version="1.0" encoding="UTF-8"?>
<n1:config xmlns:n1="http://www.bea.com/ns/wlevs/config/server"
          xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
<domain>
   <name>myDomain</name>
</domain>
...
<data-source>
  <name>derby1</name>
  <connection-pool-params>
    <initial-capacity>15</initial-capacity>
    <max-capacity>50</max-capacity>
  </connection-pool-params>
  <driver-params>
    <url>jdbc:derby:dbtest1;create=true</url>
    <driver-name>org.apache.derby.jdbc.EmbeddedDriver</driver-name>
  </driver-params>
</data-source>
</n1:config>
```

You can, of course, name the data source anything you want, as well as specify different connection pool parameters and database name (the sample above connects to the database dbtest1 and specifies that the database should be created if it does not already exist). The only element that must be the same as in the sample above is <driver-name>, whose value must be org.apache.derby.jdbc.EmbeddedDriver. You are required to add a <transaction-manager> element, although you can name it anything you want.

4. Add a transaction-manager element to the config.xml file:

```
<data-source>
 ...
</data-source>
<transaction-manager>
  <name>myTransactionManager</name>
</transaction-manager>
```

5. Add an rdbms-event-store-provider element to the config.xml file that in turn references the previously configured data source.

For example, to use the Apache Derby database configured in the previous step, add the following XML snippet:

```
<data-source>
  <name>derby1</name>
  ...
</data-source>
<rdbms-event-store-provider>
   <name>my-rdbms-provider</name>
   <data-source-name>derby1</data-source-name>
</rdbms-event-store-provider>
```

Later, when you configure the components that record and playback the events, you will specify the name of this event store.

Oracle CEP is now configured for an event store and you can configure specific components in your application to record or playback events.

13.2.2 Configuring a Component to Record Events

You can configure any processor, adapter, channel, or event bean in your application to record events. As with all other component configuration, you specify that a component records events by updating its configuration file. For general information about these configuration files, see Section 1.1.5, "Component Configuration Files."

This section describes the main steps to configure a component to record events. For simplicity, it is assumed in the procedure that you are configuring an adapter to record events and that you have already created its component configuration file.

See Section B.1, "Component Configuration Schema wlevs_application_config.xsd" for the complete XSD Schema that describes the event recording configuration file elements.

Using your favorite XML editor, open the component configuration XML file and add a `record-parameters` child element to the component you want to configure to record events. For example, to configure an adapter called `simpleEventSource`:

```xml
<?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">
    <adapter>
        <name>simpleEventSource</name>
        <record-parameters>
          ...
        </record-parameters>
        ...
    </adapter>
    ...
</n1:config>
```

Add child elements to `record-parameters` to specify the name of the event store provider, the events that are stored, the start and stop time for recording, and so on. For example:

```xml
<adapter>
        <name>simpleEventSource</name>
        <record-parameters>
          <dataset-name>recplay_sample</dataset-name>
          <event-type-list>
              <event-type>SimpleEvent</event-type>
          </event-type-list>
          <provider-name>my-rdbms-provider</provider-name>
          <batch-size>1</batch-size>
          <batch-time-out>10</batch-time-out>
        </record-parameters>
</adapter>
```

Table 13–1 lists the child elements of `record-parameters` that you can specify. Although only `dataset-name` is required, Oracle recommends that you also include `provider-name` to explicitly specify the name of the event store provider.

Table 13–1   Child Elements of record-parameters

Child Element	Description
dataset-name	Specifies the group of data that the user wants to group together.
	In the case of the Oracle RDBMS-based provider, it specifies the database area, or schema, in which the tables that store the recorded events are created.
	When configuring the Oracle RDBMS-based provider, you are required to specify this element.

Table 13–1 (Cont.) Child Elements of record-parameters

Child Element	Description
`provider-name`	Specifies the name of the event store provider.
	The value of this element corresponds to the value of the `name` child element of the `rdbms-event-store-provider` element in the `config.xml` file of the Oracle CEP server instance.
	See Section 13.2.1, "Configuring an Event Store for Oracle CEP Server."
	When configuring the Oracle RDBMS-based provider, you are required to specify this element.
`event-type-list`	Specifies the event types that are recorded to the event store. If this element is not specified, then Oracle CEP records *all* event types that flow out of the component.
	Use the `event-type` child component to list all events, such as:
	``` <event-type-list> <event-type>EventOne</event-type> <event-type>EventTwo</event-type> </event-type-list> ```
	When configuring the Oracle RDBMS-based provider, you are required to specify this element.
`time-range`	Specifies the time period during which recording should take place using a start and end time.
	The time period is configured by using a `start` child element to specify a start time and an `end` child element to specify the end time. The start and end time formats are both `MM-dd-yyyy:HH:mm:ss`, such as `10-20-2007:11:22:07`. For example, to specify that recording should start on April 7, 2009, at 6:00am and end on April 10, 2009, at 10:00 pm, enter the following
	``` <time-range-offset> <start>04-07-2009:06:00:00</start> <end>04-10-2009:22:00:00</end> </time-range-offset> ```
	If you do not specify a time period, then no events are recorded when the application is deployed and recording will only happen after you explicitly start it using Oracle CEP Visualizer or `wlevs.Admin`.
	You can specify `time-range` or `time-range-offset`, but not both.
`time-range-offset`	Specifies the time period during which recording should take place, using a start time and a duration.
	The time period is configured by using a `start` child element to specify a start time and `duration` child element to specify the amount of time after the start time that recording should stop. The start time format is `MM-dd-yyyy:HH:mm:ss`, such as `10-20-2007:11:22:07`. The duration format is `HH:mm:ss`, such as `01:00:00`. For example, to specify that recording should start on April 7, 2009, at 6:00am and continue for 3 hours, enter the following
	``` <time-range-offset> <start>04-07-2009:06:00:00</start> <duration>03:00:00</duration> </time-range-offset> ```
	If you do not specify a time period, then no events are recorded when the application is deployed and recording will only happen after you explicitly start it using Oracle CEP Visualizer or `wlevs.Admin`.
	You can specify `time-range` or `time-range-offset`, but not both.
`batch-size`	Specifies the number of events that Oracle CEP picks up in a single batch from the event buffer to write the event store.
	Default value is 1000.
`batch-time-out`	Specifies the number of seconds that Oracle CEP waits for the event buffer window to fill up with the `batch-size` number of events before writing to the event store.
	Default value is 60

Table 13–1   (Cont.) Child Elements of record-parameters

Child Element	Description
max-size	If specified, Oracle CEP uses a stream when writing to the event store, and this element specifies the size of the stream, with non-zero values indicating asynchronous writes. Default value is 1024.
max-threads	If specified, Oracle CEP uses a stream when writing to the event store, and this element specifies the maximum number of threads that will be used to process events for this stream. Setting this value has no effect when max-size is 0. The default value is 1.
store-policy-parameters	Specifies policy parameters, specific to the event store provider. Use the \<name\> and \<value\> child elements to specify a particular parameter, such as: ```\n<store-policy-parameters>\n <name>supportsTransactions</name>\n <value>true</value>\n</store-policy-parameters>\n```

13.2.3 Configuring a Component to Playback Events

You can configure any processor, adapter, channel, or event bean in your application to playback events, although the component must be a node downstream of the recording component so that the playback component will actually receive the events and play them back. As with all other component configuration, you specify that a component plays back events by updating its configuration file. For general information about these configuration files, see Section 1.1.5, "Component Configuration Files."

This section describes the main steps to configure a component to play back events. For simplicity, it is assumed in the procedure that you are configuring a channel to playback events from a node upstream in the EPN that has recorded events, and that you have already created the channel's configuration file.

See for the complete XSD Schema that describes the event playback configuration file elements.

Using your favorite XML editor, open the component configuration XML file and add a `playback-parameters` child element to the component you want to configure to playback events. For example, to configure a channel called `eventStream`:

```
<?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">
    <channel>
      <name>eventStream</name>
      <playback-parameters>
        ...
      </playback-parameters>
    </channel>
    ...
</n1:config>
```

Add child elements to `playback-parameters` to specify the name of the event store provider, the events that are played back, and so on. For example:

```
    <channel>
      <name>eventStream</name>
      <playback-parameters>
          <dataset-name>recplay_sample</dataset-name>
          <event-type-list>
              <event-type>SimpleEvent</event-type>
```

```
        </event-type-list>
        <provider-name>test-rdbms-provider</provider-name>
      </playback-parameters>
    </channel>
```

Table 13–2 lists the child elements of `playback-parameters` that you can specify. Although only `dataset-name` is required, Oracle recommends that you also include `provider-name` to explicitly specify the name of the event store provider.

Table 13–2   Child Elements of playback-parameters

Child Element	Description
`dataset-name`	Specifies the group of data that the user wants to group together.
	In the case of the Oracle RDBMS-based provider, it specifies the database area, or schema, in which the tables that store the recorded events are queried for the playback events.
	When configuring the Oracle RDBMS-based provider, you are required to specify this element.
`provider-name`	Specifies the name of the event store provider.
	The value of this element corresponds to the value of the `name` child element of the `rdbms-event-store-provider` element in the `config.xml` file of the Oracle CEP server instance.
	See Section 13.2.1, "Configuring an Event Store for Oracle CEP Server."
	When configuring the Oracle RDBMS-based provider, you are required to specify this element.
`event-type-list`	Specifies the event types that are played back from the event store. If this element is not specified, then Oracle CEP plays back *all* event types.
	Use the `event-type` child component to list all events, such as:

	`<event-type-list>`
	`    <event-type>EventOne</event-type>`
	`    <event-type>EventTwo</event-type>`
	`</event-type-list>`

	When configuring the Oracle RDBMS-based provider, you are required to specify this element.
`time-range`	Specifies the time period of the recorded events that you want to play back. In other words, this element acts as a filter so that only the events recorded during the specified time period are played back. This element does *not* specify a time period when the playback itself occurs.
	The time period is configured by using a `start` child element to specify a start time and an `end` child element to specify the end time. The start and end time formats are both `MM-dd-yyyy:HH:mm:ss`, such as `10-20-2007:11:22:07`. For example, to specify that you want to playback events that were recorded starting on April 7, 2009, at 6:00am and ending on April 10, 2009, at 10:00 pm, enter the following

	`<time-range-offset>`
	`  <start>04-07-2009:06:00:00</start>`
	`  <end>04-10-2009:22:00:00</end>`
	`</time-range-offset>`

	If you do not specify a time period, then all events are played back.
	You can specify `time-range` or `time-range-offset`, but not both.

Table 13–2 (Cont.) Child Elements of playback-parameters

Child Element	Description
time-range-offset	Specifies the time period of the recorded events that you want to play back. In other words, this element acts as a filter so that only the events recorded during the specified time period are played back. This element does *not* specify a time period when the playback itself occurs.
	The time period is configured by using a `start` child element to specify a start time and `duration` child element to specify an amount of time after the start time. The start time format is `MM-dd-yyyy:HH:mm:ss`, such as `10-20-2007:11:22:07`. The duration format is `HH:mm:ss`, such as `01:00:00`. For example, to specify that you want to playback events that were recorded starting on April 7, 2009, at 6:00am and continued for three hours enter the following:
	``` <time-range-offset> <start>04-07-2009:06:00:00</start> <duration>03:00:00</duration> </time-range-offset> ```
	If you do not specify a time period, then all events are played back.
playback-speed	Specifies the playback speed as a positive float.
	The default value is 1, which corresponds to normal speed. A value of 2 means that events will be played back 2 times faster than the original record speed. Similarly, a value of 0.5 means that events will be played back at half the speed.
repeat	Specifies whether to playback events again after the playback of the specified time interval is over.
	Valid values are `true` and `false`. Default value is `false`. A value of `true` means that the repeat of playback continues an infinite number of times until it is deliberately stopped. `False` means that events will be played back only once.
max-size	If specified, Oracle CEP uses a stream when playing back events from the event store, and this element specifies the size of the stream, with non-zero values indicating asynchronous writes.
	Default value is 1024.
max-threads	If specified, Oracle CEP uses a stream when playing back events from the event store, and this element specifies the maximum number of threads that will be used to process events for this stream. Setting this value has no effect when `max-size` is 0.
	The default value is 1.
store-policy-parameters	Specifies policy parameters, specific to the event store provider. Use the `<name>` and `<value>` child elements to specify a particular parameter, such as:
	``` <store-policy-parameters> <name>supportsTransactions</name> <value>true</value> </store-policy-parameters> ```

13.2.4 Starting and Stopping the Record and Playback of Events

After you configure the record and playback functionality for the components of an application, and you deploy the application to Oracle CEP, the server starts to record events only if you specified an explicit start/stop time in the initial configuration.

For example, if you included the following element in a component configuration:

```
<time-range-offset>
    <start>04-07-2009:06:00:00</start>
    <end>04-10-2009:22:00:00</end>
</time-range-offset>
```

then recording will automatically start on April 7, 2009.

The only way to start the playback of events, however, is by using Oracle CEP Visualizer or `wlevs.Admin`. You also use these tools to dynamically start and stop the recording of events.

For more information, see:

- "Recording and Playing Back Events Flowing Through an EPN" in the *Oracle CEP Visualizer User's Guide*

- "Commands for Controlling Event Record and Playback" in the *Oracle CEP Administrator's Guide*

Visualizer and `wlevs.Admin` use managed beans (MBeans) to dynamically start and stop event recording and playback, as well as manage the event store configuration. A managed bean is a Java bean that provides a Java Management Extensions (JMX) interface. JMX is the Java EE solution for monitoring and managing resources on a network. You can create your own administration tool and use JMX to manage event store functionality by using the `com.bea.wlevs.management.configuration.StageMBean`.

For more information, see:

- "Configuring JMX for Oracle CEP" in the *Oracle CEP Administrator's Guide*

- *Oracle CEP Java API Reference*

13.2.5 Description of the Database Tables Created by the RDMBS Provider

When you enable event recording for a component, Oracle CEP automatically creates, if they do not already exist, the database tables that stores the actual data. The server connects to the database using the data source configured for the event store provider.

Oracle recommends that you allow Oracle CEP to create and manage these tables; however, if you want more control you can create your own tables.

The following guidelines describe how Oracle CEP creates the database tables, and thus the requirements if you decide to create your own tables. If the default name of a database object (schema, table, column) is a reserved word, Oracle CEP adds a suffix to come up with a non-reserved word.

- Events are stored in a schema, whose name is specified by the value of the `dataset-name` element.

- Every Java event type in the event store has a corresponding table in the schema. The name of the table is same as the name of the event type.

- Every field in the Java event type has a corresponding column in the table. The name of the column is the same as the name of the field.

  See Table 13–3 for the list of supported Java types for the fields of the event type.

- The table contains an additional column `recordTime` that stores the time the event was recorded.

- The table has an index on the `recordTime` column.

Table 13–3   Supported Java Types

Java Type	Corresponding Java SQL Type
`boolean, java.lang.Boolean`	`java.sql.Types.BIT`
`byte, java.lang.Byte`	`java.sql.Types.TINYINT`
`short, java.lang,Short`	`java.sql.Types.SMALLINT`

Table 13–3   (Cont.) Supported Java Types

Java Type	Corresponding Java SQL Type
int, java.lang.Integer	java.sql.Types.INTEGER
long, java.lang.Long	java.sql.Types.BIGINT
float, java.lang.Float	java.sql.Types.REAL
double, java.lang.Double	java.sql.Types.DOUBLE
java.lang.String	java.sql.Types.VARCHAR
java.math.BigDecimal	java.sql.Types.BIGINT
java.sql.Date, java.util.Date	java.sql.Types.TIMESTAMP
java.sql.Time	java.sql.Types.TIME
java.sql.Timestamp	java.sql.Types.TIMESTAMP

13.3 Creating a Custom Event Store Provider

Oracle CEP provides an event store API that you can use to create a custom event store provider. Oracle provides an RDBMS-based implementation for storing events in a relational database, or one that supports JDBC connections. If you want to store events in a different kind of database, or for some reason the Oracle RDBMS provider is not adequate for your needs, then you can create your own event store provider using the event store API.

The event store API is in the `com.bea.wlevs.eventstore` package; the following list describes the most important interfaces:

- `EventStore`—Object that represents a single event store. The methods of this interface allow you to persist events to the store and to query the contents of the store using a provider-specific query.

- `EventStoreManager`—Manages event stores. Only one instance of the `EventStoreManager` ever exists on a given Oracle CEP server, and this instance registers itself in the OSGI registry so that event store providers can in turn register themselves with the event store manager. You use this interface to find existing event stores, create new ones, get the provider for a given event store, and register an event provider. The event store manager delegates the actual work to the event store provider.

- `EventStoreProvider`—Underlying repository that provides event store services to clients.

For more information, see the *Oracle CEP Java API Reference*.

Part IV

Assembly, Deployment, and Testing

Part IV contains the following chapters:

14

Assembling and Deploying Oracle CEP Applications

This section contains information on the following subjects:

- Section 14.1, "Overview of Application Assembly and Deployment"
- Section 14.2, "Assembling an Oracle CEP Application"
- Section 14.3, "Deploying Oracle CEP Applications"

14.1 Overview of Application Assembly and Deployment

The term *application assembly* refers to the process of packaging the components of an application, such as the Java files and XML configuration files, into an OSGI bundle that can be deployed to Oracle CEP. The term *application deployment* refers to the process of making an application available for processing client requests in an Oracle CEP domain.

In the context of Oracle CEP assembly and deployment, an application is defined as an OSGi bundle (see
`http://www2.osgi.org/javadoc/r4/org/osgi/framework/Bundle.html`)
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.

- One or more Oracle CEP configuration XML files that configure the components of the application. The only type of component that is required to have a configuration file is the complex event processor; all other components (adapters and streams) do not require configuration files if the default configuration of the component is adequate. You can combine all configuration files into a single file, or separate the configuration for individual components in their own files.

  The configuration files must be located in the `META-INF/wlevs` directory of the OSGi bundle JAR file if you plan to dynamically deploy the bundle. If you have an application already present in the domain directory, then the configuration files need to be extracted in the same directory.

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

  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.

The OSGI bundle declares dependencies by specifying imported and required packages. It also provides functionality to other bundles by exporting packages. If a bundle is required to provide functionality to other bundles, you must use `Export-Package` to allow other bundles to reference named packages. All packages not exported are not available outside the bundle.

See Section 14.2, "Assembling an Oracle CEP Application" for detailed instructions on creating this deployment bundle.

After you have assembled the application, you deploy it by making it known to the Oracle CEP domain using the Deployer utility (packaged in the `wlevsdeploy.jar` file). For detailed instructions, see Section 14.3, "Deploying Oracle CEP Applications."

Once the application is deployed to Oracle CEP, the configured adapters immediately start listening for events for which they are configured, such as financial data feeds and so on.

> **Note:** Oracle CEP applications are built on top of the Spring Framework and OSGi Service Platform and make extensive use of their technologies and services. See Appendix A, "Additional Information about Spring and OSGi," for links to reference and conceptual information about Spring and OSGi.

14.2 Assembling an Oracle CEP Application

Assembling an Oracle CEP application refers to bundling the artifacts that make up the application into an OSGi bundle JAR file as `http://www2.osgi.org/javadoc/r4/org/osgi/framework/Bundle.html` describes. These artifacts include:

- compiled Java classes
- Oracle CEP component configuration files that configure application components (such as the processors or adapters)
- EPN assembly file
- `MANIFEST.MF` file

See Appendix A, "Additional Information about Spring and OSGi," for links to reference and conceptual information about Spring and OSGi.

14.2.1 Assembling an Oracle CEP Application Using Oracle CEP IDE for Eclipse

You can use Oracle CEP IDE for Eclipse to easily assemble your Oracle CEP application.

For more information, see:

- Section 3.3, "Exporting Oracle CEP Projects"
- Section 3.4, "Upgrading Projects"
- Section 3.5, "Managing Oracle CEP Project Libraries"

14.2.2 Assembling an Oracle CEP Application Manually

Optionally, you can assemble your Oracle CEP application manually.

For simplicity, the following procedure creates a temporary directory that contains the required artifacts, and then jars up the contents of this temporary directory. This is just

a suggestion and you are not required, of course, to assemble the application using this method.

> **Note:** See the HelloWorld example source directory for a sample `build.xml` Ant file that performs many of the steps described below. The `build.xml` file is located in *ORACLE_CEP_HOME*\ocep_ 11.1\samples\source\applications\helloworld, where *ORACLE_CEP_HOME* refers to the main Oracle CEP installation directory, such as d:\oracle_cep.

To assemble an Oracle CEP application manually:

1. Open a command window and set your environment as described in "Setting Your Development Environment" in the *Oracle CEP Getting Started*.

2. Create an empty directory, such as `output`:

   ```
   prompt> mkdir output
   ```

3. Compile all application Java files into the `output` directory.

4. Create an `output/META-INF/spring` directory.

5. Copy the EPN assembly file that describes the components of your application and how they are connected into the `output/META-INF/spring` directory.

   See Section 1.4, "Creating the EPN Assembly File" for details about this file.

6. Create an `output/META-INF/wlevs` directory.

7. Copy the XML files that configure the components of your application (such as the processors or adapters) into the `output/META-INF/wlevs` directory.

   You create these XML files during the course of creating your application, as described in Section 1.1, "Overview of the Oracle CEP Programming Model."

8. Create a `MANIFEST.MF` file that contains descriptive information about the bundle.

   See Section 14.2.2.1, "Creating the MANIFEST.MF File."

9. If you need to access third-party JAR files from your Oracle CEP application, see Section 14.2.2.2, "Accessing Third-Party JAR Files."

10. Create a JAR file that contains the contents of the `output` directory.

    Be sure you specify the `MANIFEST.MF` file you created in the previous step rather than the default manifest file.

    You can name the JAR file anything you want. In the Oracle CEP examples, the name of the JAR file is a combination of Java package name and version, such as:

    ```
    com.bea.wlevs.example.helloworld_1.0.0.0.jar
    ```

    Consider using a similar naming convention to clarify which bundles are deployed to the server.

    See the Apache Ant documentation at http://ant.apache.org/manual/CoreTasks/jar.html for information on using the `jar` task or the J2SE documentation at http://java.sun.com/j2se/1.5.0/docs/tooldocs/windows/jar.html for information on using the `jar` command-line tool.

14.2.2.1 Creating the MANIFEST.MF File

The structure and contents of the `MANIFEST.MF` file is specified by the OSGi Framework. Although the value of many of the headers in the file is specific to your application or business, many of the headers are required by Oracle CEP.

In particular, the `MANIFEST.MF` file defines the following:

- Application name—Specified with the `Bundle-Name` header.

- Symbolic application name—Specified with the `Bundle-SymbolicName` header.

  Many of the Oracle CEP tools, such as the `wlevs.Admin` utility and JMX subsystem, use the symbolic name of the bundle when referring to the application.

- Application version—Specified with the `Bundle-Version` header.

- Imported packages—Specified with the `Import-Package` header.

  Oracle CEP requires that you import the following packages at a minimum:

  ```
  Import-Package:
   com.bea.wlevs.adapter.defaultprovider;version="2.0.0.0",
   com.bea.wlevs.ede;version="2.0.0.0",
   com.bea.wlevs.ede.api;version="2.0.0.0",
   com.bea.wlevs.ede.impl;version="2.0.0.0",
   org.osgi.framework;version="1.3.0",
   org.springframework.beans.factory;version="2.0.5",
   org.apache.commons.logging;version="1.1.0",
   com.bea.wlevs.spring;version="2.0.0.0",
   com.bea.wlevs.util;version="2.0.0.0",
   org.springframework.beans;version="2.0.5",
   org.springframework.util;version="2.0",
   org.springframework.core.annotation;version="2.0.5",
   org.springframework.beans.factory;version="2.0.5",
   org.springframework.beans.factory.config;version="2.0.5",
   org.springframework.osgi.context;version="1.0.0",
   org.springframework.osgi.service;version="1.0.0"
  ```

  If you have extended the configuration of an adapter, then you must also import the following packages:

  ```
  javax.xml.bind;version="2.0",
  javax.xml.bind.annotation;version=2.0,
  javax.xml.bind.annotation.adapters;version=2.0,
  javax.xml.bind.attachment;version=2.0,
  javax.xml.bind.helpers;version=2.0,
  javax.xml.bind.util;version=2.0,
  com.bea.wlevs.configuration;version="2.0.0.0",
  com.bea.wlevs.configuration.application;version="2.0.0.0",
  com.sun.xml.bind.v2;version="2.0.2"
  ```

- Exported packages—Specified with the `Export-Package` header. You should specify this header only if you need to share one or more application classes with other deployed applications. A typical example is sharing an event type JavaBean.

  If possible, you should export packages that include only the interfaces, and not the implementation classes themselves. If other applications are using the exported classes, you will be unable to fully undeploy the application that is exporting the classes.

  Exported packages are server-wide, so be sure their names are unique across the server.

The following complete `MANIFEST.MF` file is from the HelloWorld example, which extends the configuration of its adapter:

```
Manifest-Version: 1.0
Archiver-Version:
Build-Jdk: 1.5.0_06
Extension-Name: example.helloworld
Specification-Title: 1.0.0.0
Specification-Vendor: Oracle.
Implementation-Vendor: Oracle.
Implementation-Title: example.helloworld
Implementation-Version: 1.0.0.0
Bundle-Version: 2.0.0.0
Bundle-ManifestVersion: 1
Bundle-Vendor: Oracle.
Bundle-Copyright: Copyright (c) 2006 by Oracle.
Import-Package: com.bea.wlevs.adapter.defaultprovider;version="2.0.0.0",
 com.bea.wlevs.ede;version="2.0.0.0",
 com.bea.wlevs.ede.impl;version="2.0.0.0",
 com.bea.wlevs.ede.api;version="2.0.0.0",
 org.osgi.framework;version="1.3.0",
 org.apache.commons.logging;version="1.1.0",
 com.bea.wlevs.spring;version="2.0.0.0",
 com.bea.wlevs.util;version="2.0.0.0",
 net.sf.cglib.proxy,
 net.sf.cglib.core,
 net.sf.cglib.reflect,
 org.aopalliance.aop,
 org.springframework.aop.framework;version="2.0.5",
 org.springframework.aop;version="2.0.5",
 org.springframework.beans;version="2.0.5",
 org.springframework.util;version="2.0",
 org.springframework.core.annotation;version="2.0.5",
 org.springframework.beans.factory;version="2.0.5",
 org.springframework.beans.factory.config;version="2.0.5",
 org.springframework.osgi.context;version="1.0.0",
 org.springframework.osgi.service;version="1.0.0",
 javax.xml.bind;version="2.0",
 javax.xml.bind.annotation;version=2.0,
 javax.xml.bind.annotation.adapters;version=2.0,
 javax.xml.bind.attachment;version=2.0,
 javax.xml.bind.helpers;version=2.0,
 javax.xml.bind.util;version=2.0,
 com.bea.wlevs.configuration;version="2.0.0.0",
 com.bea.wlevs.configuration.application;version="2.0.0.0",
 com.sun.xml.bind.v2;version="2.0.2"
Bundle-Name: example.helloworld
Bundle-Description: WLEvS example helloworld
Bundle-SymbolicName: helloworld
```

14.2.2.2 Accessing Third-Party JAR Files

When creating your Oracle CEP applications, you might need to access legacy libraries within existing third-party JAR files. There are two ways to ensure access to this legacy code:

- Section 14.2.2.2.1, "Accessing Third-Party JAR Files Using Bundle-Classpath"
- Section 14.2.2.2.2, "Accessing Third-Party JAR Files Using -Xbootclasspath/a"

14.2.2.2.1 Accessing Third-Party JAR Files Using Bundle-Classpath The recommended approach is to package the third-party JAR files in your Oracle CEP application JAR file. You can put the JAR files anywhere you want.

However, to ensure that your Oracle CEP application finds the classes in the third-party JAR file, you must update the application classpath by adding the `Bundle-Classpath` header to the `MANIFEST.MF` file. Set `Bundle-Classpath` to a comma-separate list of the JAR file path names that should be searched for classes and resources. Use a period (`.`) to specify the bundle itself. For example:

```
Bundle-Classpath: ., commons-logging.jar, myExcitingJar.jar,
myOtherExcitingJar.jar
```

If you need to access native libraries, you must also package them in your JAR file and use the `Bundle-NativeCode` header of the `MANIFEST.MF` file to specify their location in the JAR.

14.2.2.2.2   Accessing Third-Party JAR Files Using -Xbootclasspath/a If the JAR files include libraries used by *all* applications deployed to Oracle CEP, such as JDBC drivers, you can add the JAR file to the server's boot classpath by specifying the `-Xbootclasspath/a` option to the `java` command in the scripts used to start up an instance of the server.

The name of the server start script is `startwlevs.cmd` (Windows) or `startwlevs.sh` (UNIX), and the script is located in the server directory of your domain directory. The out-of-the-box sample domains are located in *ORACLE_CEP_HOME*`/ocep_11.1/samples/domains`, and the user domains are located in *ORACLE_CEP_HOME*`/user_projects/domains`, where *ORACLE_CEP_HOME* refers to the main Oracle CEP installation directory, such as `d:\oracle_cep`.

Update the start script by adding the `-Xbootclasspath/a` option to the `java` command that executes the `wlevs_2.0.jar` file. Set the `-Xbootclasspath/a` option to the full pathname of the third-party JAR files you want to access system-wide.

For example, if you want all deployed applications to be able to access a JAR file called `e:\jars\myExcitingJAR.jar`, update the `java` command in the start script as follows (updated section shown in bold):

```
  %JAVA_HOME%\bin\java -Dwlevs.home=%USER_INSTALL_DIR% -Dbea.home=%BEA_HOME%
-Xbootclasspath/a:e:\jars\myExcitingJAR.jar -jar "%USER_INSTALL_DIR%\bin\wlevs_
2.0.jar" -disablesecurity %1 %2 %3 %4 %5 %6
```

14.3 Deploying Oracle CEP Applications

After you assemble your Oracle CEP application, you deploy it to an Oracle CEP server domain.

This section describes:

- Section 14.3.1, "How to Deploy an Oracle CEP Application Using Oracle CEP IDE for Eclipse"

- Section 14.3.2, "How to Deploy an Oracle CEP Application Using Oracle CEP Visualizer"

- Section 14.3.3, "How to Deploy an Oracle CEP Application Using the Deployer Utility"

For more information, see:

- "Deploying an Oracle CEP Application to a Standalone-Server Domain" in the *Oracle CEP Administrator's Guide*

■ "Deploying an Oracle CEP Application to a Multi-Server Domain" in the *Oracle CEP Administrator's Guide*

14.3.1 How to Deploy an Oracle CEP Application Using Oracle CEP IDE for Eclipse

You can deploy an Oracle CEP application using Oracle CEP IDE for Eclipse.

Using the Oracle CEP IDE for Eclipse, you can deploy an application to a stand-alone domain. To deploy an application to a multi-server domain, see:

■ Section 14.3.2, "How to Deploy an Oracle CEP Application Using Oracle CEP Visualizer"

■ Section 14.3.3, "How to Deploy an Oracle CEP Application Using the Deployer Utility"

To deploy an Oracle CEP application using Oracle CEP IDE for Eclipse:

1. Assemble your Oracle CEP application.

   See Section 14.2, "Assembling an Oracle CEP Application."

2. Use the Oracle CEP IDE for Eclipse to deploy your application.

   See Section 4.3.5, "How to Deploy an Application to an Oracle CEP Server".

14.3.2 How to Deploy an Oracle CEP Application Using Oracle CEP Visualizer

The simplest way to deploy an Oracle CEP application to an Oracle CEP server domain is to use the Oracle CEP Visualizer.

Using the Oracle CEP Visualizer, you can deploy an application to either a stand-alone or multi-server domain.

To deploy an Oracle CEP application using Oracle CEP Visualizer:

1. Assemble your Oracle CEP application.

   See Section 14.2, "Assembling an Oracle CEP Application."

2. Start the Oracle CEP Visualizer.

   See Section 4.3.8, "How to Start the Oracle CEP Visualizer from Oracle CEP IDE for Eclipse".

3. Use the Oracle CEP Visualizer to deploy your application.

   See "Deploying an Application" in the *Oracle CEP Visualizer User's Guide*.

14.3.3 How to Deploy an Oracle CEP Application Using the Deployer Utility

The following procedure describes how to deploy an application to Oracle CEP using the Deployer command-line utility.

Using the Deployer, you can deploy an application to either a stand-alone or multi-server domain.

For more information, see "Deployer Command-Line Reference" in the *Oracle CEP Administrator's Guide*.

To deploy an Oracle CEP application using the Deployer utility:

1. Assemble your Oracle CEP application.

   See Section 14.2, "Assembling an Oracle CEP Application."

2. Open a command window and set your environment as described in "Setting Your Development Environment" in the *Oracle CEP Getting Started*.

3. Update your `CLASSPATH` variable to include the `wlevsdeploy.jar` JAR file, located in the *ORACLE_CEP_HOME*/`ocep_11.1/bin` directory where, *ORACLE_ CEP_HOME* refers to the main Oracle CEP installation directory, such as `/oracle_ cep`.

> **Note:** If you are running the Deployer on a remote computer, see "Running the Deployer Utility Remotely" in the *Oracle CEP Administrator's Guide*.

4. Be sure you have configured Jetty for the Oracle CEP instance to which you are deploying your application.

For more information, see "Configuring Jetty for Oracle CEP" in the *Oracle CEP Administrator's Guide*.

5. In the command window, run the Deployer utility using the following syntax to install your application:

```
prompt> java -jar wlevsdeploy.jar -url http://host:port/wlevsdeployer -user
user -password password -install application_jar_file
```

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; default value is `9002`.

  This port is specified in the *DOMAIN_DIR*/`config/config.xml` file that describes your Oracle CEP domain, 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>
```

- *user* refers to the username of the Oracle CEP administrator.

- *password* refers to the password of the Oracle CEP administrator.

- *application_jar_file* refers to your application JAR file, assembled into an OSGi bundle as described in Section 14.2, "Assembling an Oracle CEP Application." This file must be located on the same computer from which you execute the Deployer utility.

  For example, if Oracle CEP is running on host `ariel`, listening on port `9002`, username and password of the administrator is `wlevs/wlevs`, and your application JAR file is called `myapp_1.0.0.0.jar` and is located in the `/applications` directory, then the command is:

```
prompt> java -jar wlevsdeploy.jar -url http://ariel:9002/wlevsdeployer
-user wlevs -password wlevs -install /applications/myapp_1.0.0.0.jar
```

After the application JAR file has been successfully installed and all initialization tasks completed, Oracle CEP automatically starts the application and the adapter components immediately start listening for incoming events.

The Deployer utility provides additional options to resume, suspend, update, and uninstall an application JAR file, as well as deploy an application to a specified group of a multi-server domain. For more information, see "Deployer Command-Line Reference" in the *Oracle CEP Administrator's Guide*.

Oracle CEP uses the `deployments.xml` file to internally maintain its list of deployed application OSGi bundles. This file is located in the *DOMAIN_ DIR/servername* directory, where *DOMAIN_DIR* refers to the main domain directory corresponding to the server instance to which you are deploying your application and *servername* refers to the actual server. See Appendix B.3, "Deployment Schema deployment.xsd" for information about this file. This information is provided for your information only; Oracle does not recommend updating the `deployments.xml` file manually.

15

Testing Applications With the Load Generator and csvgen Adapter

This section contains information on the following subjects:

15.1 Overview of Testing Applications With the Load Generator and csvgen Adapter

The load generator is a simple utility provided by Oracle CEP to simulate a data feed. The utility is useful for testing the Oracle CQL or EPL rules in your application without needing to connect to a real-world data feed.

The load generator reads an ASCII file that contains the sample data feed information and sends each data item to the configured port. The load generator reads items from the sample data file in order and inserts them into the channel, looping around to the beginning of the data file when it reaches the end; this ensures that a continuous stream of data is available, regardless of the number of data items in the file. You can configure the rate of sent data, from the rate at which it starts, the final rate, and how long it takes the load generator to ramp up to the final rate.

In your application, you must use the Oracle CEP-provided `csvgen` adapter, rather than your own adapter, to read the incoming data; this is because the `csvgen` adapter is specifically coded to decipher the data packets generated by the load generator.

If you redeploy your application, you must also restart the load generator.

15.2 Configuring and Running the Load Generator Utility

This procedure describes how to configure and run the load generator utility.

To configure and run the load generator utility:

1. Optionally create a property file that contains configuration properties for particular run of the load generator; these properties specify the location of the file

that contains simulated data, the port to which the generator feeds the data, and so on.

Oracle CEP provides a default property file you can use if the default property values are adequate.

See Section 15.3, "Creating a Load Generator Property File."

2. Create a file that contains the actual data feed values.

See Section 15.4, "Creating a Data Feed File."

3. Configure the `csvgen` adapter so that it correctly reads the data feed generated by the load generator. You configure the adapter in the EPN assembly file that describes your Oracle CEP application.

See Section 15.5, "Configuring the csvgen Adapter in Your Application."

4. Be sure that you configure a builder factory for creating your event types. Although specifying event type builder factories is typically an optional task, it is required when using the load generator.

See Section 1.5, "Creating Oracle CEP Event Types" for details.

5. Open a command window and set your environment as described in "Setting Your Development Environment" in the *Oracle CEP Getting Started*.

6. Change to the `ORACLE_CEP_HOME\ocep_11.1\utils\load-generator` directory, where `ORACLE_CEP_HOME` refers to the main Oracle CEP installation directory, such as `d:\oracle_cep`.

7. Run the load generator specifying the properties file you created in step 1 to begin the simulated data feed. For example, if the name of your properties file is `c:\loadgen\myDataFeed.prop`, execute the following command:

```
prompt> runloadgen.cmd c:\loadgen\myDataFeed.prop
```

15.3 Creating a Load Generator Property File

The load generator uses an ASCII properties file for its configuration purposes. Properties include the location of the file that contains the sample data feed values, the port to which the utility should send the data feed, and so on.

Oracle CEP provides a default properties file called `csvgen.prop`, located in the `ORACLE_CEP_HOME\ocep_11.1\utils\load-generator` directory, where `ORACLE_CEP_HOME` refers to the main Oracle CEP installation directory, such as `d:\oracle_cep`.

The format of the file is simple: each property-value pair is on its own line. The following example shows the default `csvgen.prop` file; Oracle recommends you use this file as a template for your own property file:

```
test.csvDataFile=test.csv
test.port=9001
test.packetType=CSV
test.mode=client
test.senders=1
test.latencyStats=false
test.statInterval=2000
```

> **Caution:** If you create your own properties file, you must include the `test.packetType`, `test.mode`, `test.senders`, `test.latencyStats`, and `test.statInterval` properties exactly as shown above.

In the preceding sample properties file, the file that contains the sample data is called `test.csv` and is located in the same directory as the properties file. The load generator will send the data feed to port `9001`.

The following table lists the additional properties you can set in your properties file.

Table 15–1   Load Generator Properties

Property	Description	Data Type	Required?
`test.csvDataFile`	Specifies the file that contains the data feed values.	`String`	Yes
`test.port`	The port number to which the load generator should send the data feed.	`Integer`	Yes
`test.secs`	Total duration of the load generator run, in seconds. The default value is 30.	`Integer`	No
`test.rate`	Final data rate, in messages per second. The default value is 1.	`Integer`	No
`test.startRate`	Initial data rate, in messages per second. The default value is 1.	`Integer`	No
`test.rampUpSecs`	Number of seconds to ramp up from `test.startRate` to `test.rate`. The default value is 0.	`Integer`	No

15.4 Creating a Data Feed File

The file that contains the sample data feed values correspond to the event type registered for your Oracle CEP application. The file follows a simple format:

- Each item of a particular data feed is on its own line.

- Separate the fields of a data feed item with commas.

- Do not include extraneous spaces before or after the commas, unless the space is literally part of the field value.

- Include only string and numerical data in a data feed file such as integer, long, double, and float.

  For more information, see "Event Types for use With the csvgen Adapter" on page 1-7.

The following example shows a sample data feed file where each item corresponds to a person with `name`, `age`, and `birthplace` fields:

```
Lucy,23,Madagascar
Nick,44,Canada
Amanda,12,Malaysia
Juliet,43,Spain
Horatio,80,Argentina
```

15.5 Configuring the csvgen Adapter in Your Application

When using the load generator utility, you must use the `csvgen` adapter in your application because this Oracle CEP-provided adapter is specifically coded to read the data packets generated by the load generator.

You register the `csvgen` adapter using the `wlevs:adapter` element in the EPN assembly file of your application, as with all adapters. Set the `provide` attribute to `csvgen` to specify that the provider is the `csvgen` adapter, rather than your own adapter. Additionally, you must specify the following child tags:

- `wlevs:instance-property` element with `name` attribute `port` and `value` attribute *`configured_port`*, where *`configured_port`* corresponds to the value of the `test.port` property in the load generator property file. See Section 15.3, "Creating a Load Generator Property File."

- `wlevs:instance-property` element with `name` attribute `eventTypeName` and `value` attribute *`event_type_name`*, where *`event_type_name`* corresponds to the name of the event type that represents an item from the load-generated feed.

- `wlevs:instance-property` element with `name` attribute `eventPropertyNames` and `value` attribute *`ordered_list_of_properties`*, where *`ordered_list_of_properties`* lists the names of the properties in the order that the load generator sends them, and consequently the `csvgen` adapter receives them.

Before showing an example of how to configure the adapter, first assume that your application registers an event type called `PersonType` in the EPN assembly file using the `wlevs:metada` element shown below:

```
<wlevs:event-type-repository>
        <wlevs:event-type type-name="PersonType">
            <wlevs:property>
                <entry key="name" value="java.lang.String"/>
                <entry key="age" value="java.lang.Integer"/>
                <entry key="birthplace" value="java.lang.String"/>
            </wlevs:property>
        </wlevs:event-type>
</wlevs:event-type-repository>
```

This event type corresponds to the data feed file shown in Section 15.4, "Creating a Data Feed File."

To configure the csvgen adapter that receives this data, use the following `wlevs:adapter` element:

```
<wlevs:adapter id="csvgenAdapter" provider="csvgen">
   <wlevs:instance-property name="port" value="9001"/>
   <wlevs:instance-property name="eventTypeName" value="PersonType"/>
   <wlevs:instance-property name="eventPropertyNames" value="name,age,birthplace"/>
</wlevs:adapter>
```

Note how the bolded values in the adapter configuration example correspond to the `PersonType` event type registration.

If you use the `wlevs:class` element to specify your own JavaBean when registering the event type, then the `eventPropertyNames` value corresponds to the JavaBean properties. For example, if your JavaBean has a `getName()` method, then one of the properties of your JavaBean is `name`.

For more information on event types, see Section 1.5, "Creating Oracle CEP Event Types".

Part V

Oracle CEP Reference

Part V contains the following chapters:

- Appendix A, "Additional Information about Spring and OSGi"
- Appendix B, "Oracle CEP Schemas"
- Appendix C, "Schema Reference: Component Configuration wlevs_application-config.xsd"
- Appendix D, "Schema Reference: EPN Assembly spring-wlevs-v11_0_0_0.xsd"
- Appendix E, "Schema Reference: Deployment deployment.xsd"
- Appendix F, "Schema Reference: Server Configuration wlevs_server_config.xsd"
- Appendix G, "Oracle CEP Metadata Annotation Reference"
- Appendix H, "Oracle CEP IDE for Eclipse Tutorial"

A

Additional Information about Spring and OSGi

Oracle Complex Event Processing applications are built on top of the Spring Framework and OSGi Service Platform. Therefore, it is assumed that you are familiar with these technologies and how to program within the frameworks.

For additional information about Spring and OSGi, see:

- Spring Framework API 2.5:
  http://static.springframework.org/spring/docs/2.5.x/api/index.html

- The Spring Framework - Reference Documentation 2.5:
  http://static.springframework.org/spring/docs/2.5.x/reference/index.html

- Spring-OSGi Project: http://www.springframework.org/osgi

- OSGi Release 4 Service Platform Javadoc:
  http://www.springframework.org/osgi

- OSGi Release 4 Core Specification: http://www.osgi.org/osgi_technology/download_specs.asp?section=2#Release4

Oracle CEP Schemas

This section contains information on the following subjects:

B.1 Component Configuration Schema wlevs_application_config.xsd

An Oracle CEP application contains one or more component configuration files in its `META-INF/wlevs` directory. You use component configuration files to override the default configuration for Oracle CEP components such as adapters, channels, and processors.

The `wlevs_application_config.xsd` schema file describes the structure of component configuration files. This XSD schema imports the following schemas:

- `wlevs_base_config.xsd`

- `wlevs_eventstore_config.xsd`

- `wlevs_diagnostic_config.xsd`

These schema files are located in the *ORACLE_CEP_HOME*`\ocep_11.1\xsd` directory, where *ORACLE_CEP_HOME* is the main Oracle CEP installation directory, such as `d:\oracle_cep`.

For more information, see Appendix C, "Schema Reference: Component Configuration wlevs_application-config.xsd".

B.1.1 Example 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>
            <query id="helloworldRule">
                <![CDATA[ select * from helloworldInputChannel [Now] ]]>
            </query>
        </rules>
```

```
                     </processor>
                     <channel>
                         <name>helloworldInputChannel</name>
                         <max-size>10000</max-size>
                         <max-threads>2</max-threads>
                     </channel>
                     <channel>
                         <name>helloworldOutputChannel</name>
                         <max-size>10000</max-size>
                         <max-threads>2</max-threads>
                     </channel>
                 </n1:config>
```

B.2 EPN Assembly Schema spring-wlevs-v11_0_0_0.xsd

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.

The spring-wlevs-v11_0_0_0.xsd file describes the structure of EPN assembly files.

This schema file is located in the *ORACLE_CEP_HOME*\ocep_11.1\xsd directory, where *ORACLE_CEP_HOME* is the main Oracle CEP installation directory, such as d:\oracle_cep.

For more information, see Appendix D, "Schema Reference: EPN Assembly spring-wlevs-v11_0_0_0.xsd".

B.2.1 Example 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-v11_0_0_0.xsd">

    <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>

    <!-- Adapter can be created from a local class, without having to go through a adapter factory -->
    <wlevs:adapter id="helloworldAdapter" class="com.bea.wlevs.adapter.example.helloworld.HelloWorldAdapter" >
        <wlevs:instance-property name="message" value="HelloWorld - the current time is:"/>
    </wlevs:adapter>

    <wlevs:channel id="helloworldInputChannel" event-type="HelloWorldEvent" >
```

```
      <wlevs:listener ref="helloworldProcessor"/>
      <wlevs:source ref="helloworldAdapter"/>
  </wlevs:channel>

  <!-- The default processor for Oracle CEP 11.0.0.0 is CQL -->
  <wlevs:processor id="helloworldProcessor" />

  <wlevs:channel id="helloworldOutputChannel" event-type="HelloWorldEvent" advertise="true">
      <wlevs:listener>
          <bean class="com.bea.wlevs.example.helloworld.HelloWorldBean"/>
      </wlevs:listener>
      <wlevs:source ref="helloworldProcessor"/>
  </wlevs:channel>

</beans>
```

B.3 Deployment Schema deployment.xsd

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.

The `deployment.xsd` schema file describes the structure of deployment files.

This schema file is located in the *ORACLE_CEP_HOME*\ocep_11.1\xsd directory, where *ORACLE_CEP_HOME* is the main Oracle CEP installation directory, such as d:\oracle_cep.

For more information, see Appendix E, "Schema Reference: Deployment deployment.xsd".

B.3.1 Example 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_11.1.0.0.jar"/>
</beans>
```

B.4 Server Configuration Schema wlevs_server_config.xsd

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.

The `welvs_server_config.xsd` schema file describes the structure of server configuration files.

This schema file is located in the *ORACLE_CEP_HOME*`\ocep_11.1\xsd` directory, where *ORACLE_CEP_HOME* is the main Oracle CEP installation directory, such as `d:\oracle_cep`.

For more information, see Appendix F, "Schema Reference: Server Configuration wlevs_server_config.xsd".

B.4.1 Example Server Configuration XML File

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

```xml
<?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>
    <jndi-service-name>JNDI</jndi-service-name>
  </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>
```

C

Schema Reference: Component Configuration wlevs_application-config.xsd

This appendix describes the elements of the `wlevs_application_config.xsd` schema.

For more information, see:

- Section C.1, "Overview of the Oracle CEP Component Configuration Elements"
- Section B.1, "Component Configuration Schema wlevs_application_config.xsd".

C.1 Overview of the Oracle CEP Component Configuration Elements

Oracle CEP provides a number of component configuration elements that you use to define the characteristics of the of the components you declare in the EPN assembly file.

C.1.1 Element Hierarchy

The top-level Oracle CEP component configuration elements are organized into the following hierarchy:

- `config`
    - `adapter` (see Example C–1)
    - `http-pub-sub-adapter` (see Example C–2)
    - `jms-adapter` (see Example C–3)
    - `processor` (see Example C–4 and Example C–5)
    - `stream` (see Example C–6)
    - `channel` (see Example C–7)
    - `event-bean` (see Example C–8)
    - `caching-system` (see Example C–9)
    - `coherence-caching-system` (see Example C–10)
    - `diagnostic-profiles` (see Example C–11)

Example C–1   adapter Element Hierarchy

adapter
   name
   record-parameters

```
            dataset-name
            event-type-list
                event-type
            provider-name
            store-policy-parameters
                parameter
                    name
                    value
            max-size
            max-threads
            time-range
                start
                end
            time-range-offset
                start
                duration
            batch-size
            batch-time-out
        playback-parameters
            dataset-name
            event-type-list
                event-type
            provider-name
            store-policy-parameters
                parameter
                    name
                    value
            max-size
            max-threads
            time-range
                start
                end
            time-range-offset
                start
                duration
            schedule-time-range
                start
                end
            schedule-time-range-offset
                start
                duration
        symbols
            symbol
    work-manager-name
    netio
        provider-name
        num-threads
        accept-backlog
```

Example C–2   http-pub-sub-adapter Element Hierarchy

```
http-pub-sub-adapter
    name
    record-parameters
        dataset-name
        event-type-list
            event-type
        provider-name
        store-policy-parameters
            parameter
```

```
                    name
                    value
            max-size
            max-threads
            time-range
                start
                end
            time-range-offset
                start
                duration
            batch-size
            batch-time-out
        playback-parameters
            dataset-name
            event-type-list
                event-type
            provider-name
            store-policy-parameters
                parameter
                    name
                    value
            max-size
            max-threads
            time-range
                start
                end
            time-range-offset
                start
                duration
            schedule-time-range
                start
                end
            schedule-time-range-offset
                start
                duration
            symbols
                symbol
        work-manager-name
        netio
            provider-name
            num-threads
            accept-backlog
        One of:
            server-context-path
            server-url
        channel (http-pub-sub-adapter Child Element)
        event-type
        user
        One of:
            password
            encrypted-password
```

Example C–3   jms-adapter Element Hierarchy

```
jms-adapter
    name
    record-parameters
        dataset-name
        event-type-list
```

```
                                event-type
                        provider-name
                        store-policy-parameters
                            parameter
                                    name
                                    value
                        max-size
                        max-threads
                        time-range
                            start
                            end
                        time-range-offset
                            start
                            duration
                        batch-size
                        batch-time-out
                playback-parameters
                        dataset-name
                        event-type-list
                            event-type
                        provider-name
                        store-policy-parameters
                            parameter
                                    name
                                    value
                        max-size
                        max-threads
                        time-range
                            start
                            end
                        time-range-offset
                            start
                            duration
                        schedule-time-range
                            start
                            end
                        schedule-time-range-offset
                            start
                            duration
                        event-type
                jndi-provider-url
                jndi-factory
                connection-jndi-name
                One of:
                        destination-jndi-name
                        destination-name
                user
                One of:
                        password
                        encrypted-password
                work-manager
                concurrent-consumers
                message-selector
                session-ack-mode-name
                session-transacted
                delivery-mode
```

Example C–4   processor (EPL) Element Hierarchy

```
processor (EPL)
    name
    record-parameters
        dataset-name
        event-type-list
            event-type
        provider-name
        store-policy-parameters
            parameter
                name
                value
        max-size
        max-threads
        time-range
            start
            end
        time-range-offset
            start
            duration
        batch-size
        batch-time-out
    playback-parameters
        dataset-name
        event-type-list
            event-type
        provider-name
        store-policy-parameters
            parameter
                name
                value
        max-size
        max-threads
        time-range
            start
            end
        time-range-offset
            start
            duration
        schedule-time-range
            start
            end
        schedule-time-range-offset
            start
            duration
    rules
        rule
        query
        view
    database
    bindings
        binding
            params
```

Example C–5   processor (Oracle CQL) Element Hierarchy

```
processor (Oracle CQL)
    name
    record-parameters
```

```
                    dataset-name
                    event-type-list
                        event-type
                    provider-name
                    store-policy-parameters
                        parameter
                            name
                            value
                    max-size
                    max-threads
                    time-range
                        start
                        end
                    time-range-offset
                        start
                        duration
                    batch-size
                    batch-time-out
                playback-parameters
                    dataset-name
                    event-type-list
                        event-type
                    provider-name
                    store-policy-parameters
                        parameter
                            name
                            value
                    max-size
                    max-threads
                    time-range
                        start
                        end
                    time-range-offset
                        start
                        duration
                    schedule-time-range
                        start
                        end
                    schedule-time-range-offset
                        start
                        duration
            rules
                rule
                query
                view
```

Example C–6   stream Element Hierarchy

```
stream
    name
    record-parameters
        dataset-name
        event-type-list
            event-type
        provider-name
        store-policy-parameters
            parameter
                name
                value
```

```
        max-size
        max-threads
        time-range
            start
            end
        time-range-offset
            start
            duration
        batch-size
        batch-time-out
    playback-parameters
        dataset-name
        event-type-list
            event-type
        provider-name
        store-policy-parameters
            parameter
                name
                value
        max-size
        max-threads
        time-range
            start
            end
        time-range-offset
            start
            duration
        schedule-time-range
            start
            end
        schedule-time-range-offset
            start
            duration
    max-size
    max-threads
```

Example C–7   channel Element Hierarchy

```
channel
    name
    record-parameters
        dataset-name
        event-type-list
            event-type
        provider-name
        store-policy-parameters
            parameter
                name
                value
        max-threads
        max-size
        time-range
            start
            end
        time-range-offset
            start
            duration
        batch-size
        batch-time-out
```

```
playback-parameters
    dataset-name
    event-type-list
        event-type
    provider-name
    store-policy-parameters
        parameter
            name
            value
    max-size
    max-threads
    time-range
        start
        end
    time-range-offset
        start
        duration
    schedule-time-range
        start
        end
    schedule-time-range-offset
        start
        duration
max-size
max-threads
selector
heartbeat-timeout
```

Example C–8   event-bean Element Hierarchy

```
event-bean
    name
    record-parameters
        dataset-name
        event-type-list
            event-type
        provider-name
        store-policy-parameters
            parameter
                name
                value
        max-size
        max-threads
        time-range
            start
            end
        time-range-offset
            start
            duration
        batch-size
        batch-time-out
    playback-parameters
        dataset-name
        event-type-list
            event-type
        provider-name
        store-policy-parameters
            parameter
                name
```

```
                value
            max-size
            max-threads
            time-range
                start
                end
            time-range-offset
                start
                duration
            schedule-time-range
                start
                end
            schedule-time-range-offset
                start
                duration
```

Example C–9   caching-system Element Hierarchy

```
caching-system
    name
    cache
        name
        max-size
        eviction-policy
        time-to-live
        idle-time
        One of:
            write-none
            write-through
            write-behind
                work-manager-name
                    batch-size
                    buffer-size
                    buffer-write-attempts
                    buffer-write-timeout
        work-manager-name
        listeners
```

Example C–10   coherence-caching-system Element Hierarchy

```
coherence-caching-system
    name
    coherence-cache-config
    coherence-cluster-config
```

Example C–11   diagnostic-profiles Element Hierarchy

```
diagnostic-profiles
    name
    profile
        name
        enabled
        start-stage
        max-latency
            name
            collect-interval
                amount
                unit
```

```
            start-location
                application
                stage
                direction
            end-location
                application
                stage
                direction
        average-latency
            name
            collect-interval
                amount
                unit
            start-location
                application
                stage
                direction
            end-location
                application
                stage
                direction
            threshhold
        throughput
            name
            throughput-interval
                amount
                unit
            average-interval
                amount
                unit
            location
                application
                stage
                direction
```

C.1.2 Example of an Oracle CEP Component Configuration File

The following sample component configuration file from the HelloWorld application shows how to use many of the Oracle CEP elements:

```xml
<?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>
            <query id="helloworldRule">
                <![CDATA[ select * from helloworldInputChannel [Now] ]]>
            </query>
        </rules>
    </processor>
    <channel>
        <name>helloworldInputChannel</name>
        <max-size>10000</max-size>
        <max-threads>2</max-threads>
    </channel>
    <channel>
        <name>helloworldOutputChannel</name>
        <max-size>10000</max-size>
```

```
            <max-threads>2</max-threads>
        </channel>
</n1:config>
```

C.2 accept-backlog

Use this element to define the maximum number of pending connections allowed on a socket. This element is only applicable in a netio element.

C.2.1 Child Elements

The `accept-backlog` component configuration element has no child elements.

C.2.2 Attributes

The `accept-backlog` component has no attributes.

C.2.3 Example

The following example shows how to use the `accept-backlog` element in the component configuration file:

```
<netio>
        <provider-name>providerCache</provider-name>
        <num-threads>1000</num-threads>
        <accept-backlog>50</accept-backlog>
</netio>
```

C.3 adapter

Use this element to define a custom adapter component. For an HTTP publish-subscribe or JMS adapter, use the specific http-pub-sub-adapter and jms-adapter elements.

For more information, see Chapter 8, "Configuring Custom Adapters, Event Beans, and Spring Beans".

C.3.1 Child Elements

The `adapter` component configuration element supports the following child elements:

- name
- record-parameters
- playback-parameters
- symbols
- work-manager-name
- netio

C.3.2 Attributes

The `adapter` component configuration element has no attributes.

C.3.3 Example

The following example shows how to use the `adapter` element in the component configuration file:

```
<adapter>
    <name>trackdata</name>
    <symbols>
        <symbol>BEAS</symbol>
        <symbol>IBM</symbol>
    </symbols>
 </adapter>
```

In the example, the adapter's unique identifier is `trackdata`.

C.4 amount

Use this element to define the a time duration of a diagnostic profile. This element is applicable in any of the following elements:

- average-interval

- collect-interval

- throughput-interval

For more information, see "Monitoring the Throughput and Latency of a Stage or Path in the EPN" in the *Oracle CEP Visualizer User's Guide*.

C.4.1 Child Elements

The `amount` component configuration has no child elements:

C.4.2 Attributes

The `amount` component has no attributes.

C.4.3 Example

The following example shows how to use the `amount` element in the component configuration file:

```
<diagnostic-profiles>
    <name>myselfprofiles</name>
    <profile>
        <name>testProfile0</name>
        <enabled>true</enabled>
        <start-stage>MetricSubscriber</start-stage>
        <max-latency>
            <collect-interval>
                <amount>1000</amount>
                <unit>s</unit>
            </collect-interval>
            <name>testProfile0MaxLat</name>
            <start-location>
                <application>diagnostic</application>
                <stage>MetricSubscriber</stage>
                <direction>INBOUND</direction>
            </start-location>
            <end-location>
                <application>diagnostic</application>
                <stage>MonitorProcessor</stage>
                <direction>OUTBOUND</direction>
```

```
            </end-location>
        </max-latency>
    </profile>
</diagnostic-profiles>
```

C.5 application

Use this element to define the type of application Oracle CEP server applies to a foreign stage. In a diagnostic profile, this element always has a value of diagnostic.

For more information, see "Monitoring the Throughput and Latency of a Stage or Path in the EPN" in the *Oracle CEP Visualizer User's Guide*.

C.5.1 Child Elements

The application component configuration has no child elements:

C.5.2 Attributes

The application component has no attributes.

C.5.3 Example

The following example shows how to use the application element in the component configuration file:

```
<diagnostic-profiles>
    <name>myselfprofiles</name>
    <profile>
        <name>testProfile0</name>
        <enabled>true</enabled>
        <start-stage>MetricSubscriber</start-stage>
        <max-latency>
            <collect-interval>
                <amount>1000</amount>
                <unit>s</unit>
            </collect-interval>
            <name>testProfile0MaxLat</name>
            <start-location>
                <application>diagnostic</application>
                <stage>MetricSubscriber</stage>
                <direction>INBOUND</direction>
            </start-location>
            <end-location>
                <application>diagnostic</application>
                <stage>MonitorProcessor</stage>
                <direction>OUTBOUND</direction>
            </end-location>
        </max-latency>
    </profile>
</diagnostic-profiles>
```

C.6 average-interval

Use this element to define the time interval for which you want to gather metrics.

For more information, see "Monitoring the Throughput and Latency of a Stage or Path in the EPN" in the *Oracle CEP Visualizer User's Guide*.

C.6.1 Child Elements

The `average-interval` component configuration element supports the following child elements:

- amount

- unit

C.6.2 Attributes

The `average-interval` component has no attributes.

C.6.3 Example

The following example shows how to use the `average-interval` element in the component configuration file:

```
<diagnostic-profiles>
    <name>myselfprofiles</name>
    <profile>
        <name>testProfile0</name>
        <enabled>true</enabled>
        <start-stage>MetricSubscriber</start-stage>
        <throughput>
            <throughput-interval>
                <amount>100000</amount>
                <unit>MICROSECONDS</unit>
            </throughput-interval>
            <average-interval>
                <amount>100000000</amount>
                <unit>NANOSECONDS</unit>
            </average-interval>
            <location>
                <application>diagnostic</application>
                <stage>AlertEventStream</stage>
                <direction>INBOUND</direction>
            </location>
        </throughput>
    </profile>
</diagnostic-profiles>
```

C.7 average-latency

Use this element to define an average latency calculation in a diagnostic profile.

For more information, see "Monitoring the Throughput and Latency of a Stage or Path in the EPN" in the *Oracle CEP Visualizer User's Guide*.

C.7.1 Child Elements

The `average-latency` component configuration element supports the following child elements:

- name

- collect-interval

- start-location

- end-location

- threshhold

C.7.2 Attributes

The `average-latency` component has no attributes.

C.7.3 Example

The following example shows how to use the `average-latency` element in the component configuration file:

```
<diagnostic-profiles>
    <name>myselfprofiles</name>
    <profile>
        <name>testProfile0</name>
        <enabled>true</enabled>
        <start-stage>MetricSubscriber</start-stage>
        <average-latency>
            <start-location>
                <application>diagnostic</application>
                <stage>MetricSubscriber</stage>
                <direction>INBOUND</direction>
            </start-location>
            <end-location>
                <application>diagnostic</application>
                <stage>MonitorProcessor</stage>
                <direction>OUTBOUND</direction>
            </end-location>
            <threshhold>
                <amount>100</amount>
                <unit>MILLISECONDS</unit>
            </threshhold>
        </average-latency>
    </profile>
</diagnostic-profiles>
```

C.8 batch-size

Use this element to define the number of updates that are picked up from the store buffer to write back to the backing store. This element may be changed dynamically.

C.8.1 Child Elements

The `batch-size` component configuration element has no child elements.

C.8.2 Attributes

The `batch-size` component has no attributes.

C.8.3 Example

The following example shows how to use the `batch-size` element in the component configuration file:

```
<record-parameters>
    <dataset-name>tuple1</dataset-name>
    <event-type-list>
        <event-type>TupleEvent1</event-type>
    </event-type-list>
    <provider-name>test-rdbms-provider</provider-name>
    <store-policy-parameters>
        <parameter>
            <name>timeout</name>
            <value>300</value>
        <parameter>
```

```
        </store-policy-parameters>
        <batch-size>1</batch-size>
        <batch-time-out>10</batch-time-out>
</record-parameters>
```

C.9 batch-time-out

Use this element to define The number of seconds event buffer will wait to accumulate batch-size number of events before to write to the event store.

C.9.1 Child Elements

The `batch-time-out` component configuration element has no child elements.

C.9.2 Attributes

The `batch-time-out` component has no attributes.

C.9.3 Example

The following example shows how to use the `batch-time-out` element in the component configuration file:

```
<record-parameters>
    <dataset-name>tuple1</dataset-name>
    <event-type-list>
        <event-type>TupleEvent1</event-type>
    </event-type-list>
    <provider-name>test-rdbms-provider</provider-name>
    <store-policy-parameters>
        <parameter>
            <name>timeout</name>
            <value>300</value>
        <parameter>
    </store-policy-parameters>
    <batch-size>1</batch-size>
    <batch-time-out>10</batch-time-out>
</record-parameters>
```

C.10 binding

Use this element to define values for a parameterized EPL rule in an EPL processor component.

For more information, see "Parameterized Queries" in the *Oracle CEP EPL Language Reference*.

C.10.1 Child Elements

The `binding` component configuration element supports the following child elements:

- params

C.10.2 Attributes

Table C–1 lists the attributes of the `binding` component configuration element.

Table C–1   Attributes of the binding Component Configuration Element

Attribute	Description	Data Type	Required?
id	The identifier of the EPL rule to which this binding applies.	String	Yes.

C.10.3 Example

The following example shows how to use the `binding` element in the component configuration file:

```
<processor>
    <name>processor1</name>
        <record-parameters>
            <dataset-name>test1data</dataset-name>
            <event-type-list>
                <event-type>SimpleEvent</event-type>
            </event-type-list>
            <provider-name>test-rdbms-provider</provider-name>
            <batch-size>1</batch-size>
            <batch-time-out>10</batch-time-out>
        </record-parameters>
        <rules>
            <rule id="rule1"><![CDATA[
                select stockSymbol, avg(price) as percentage
                from StockTick retain 5 events
                where stockSymbol=?
                having avg(price)  > ? or avg(price)  < ?
            ]]></rule>
        </rules>
        <bindings>
            <binding id="rule1">
                <params>BEAS,10.0,-10.0</params>
                <params id="IBM">IBM,5.0,5.0</params>
            </binding>
        </bindings>
</processor>
```

C.11 bindings

Use this element to define bindings for one or more parameterized EPL rules in an EPL processor component.

For more information, see "Parameterized Queries" in the *Oracle CEP EPL Language Reference*.

C.11.1 Child Elements

The `bindings` component configuration element supports the following child elements:

- binding

C.11.2 Attributes

The `bindings` component has no attributes.

C.11.3 Example

The following example shows how to use the `bindings` element in the component configuration file:

```
<processor>
    <name>processor1</name>
        <record-parameters>
            <dataset-name>test1data</dataset-name>
            <event-type-list>
                <event-type>SimpleEvent</event-type>
            </event-type-list>
            <provider-name>test-rdbms-provider</provider-name>
            <batch-size>1</batch-size>
            <batch-time-out>10</batch-time-out>
        </record-parameters>
        <rules>
            <rule id="rule1"><![CDATA[
                select stockSymbol, avg(price) as percentage
                from StockTick retain 5 events
                where stockSymbol=?
                having avg(price)  > ? or avg(price)  < ?
            ]]></rule>
        </rules>
        <bindings>
            <binding id="rule1">
                <params>BEAS,10.0,-10.0</params>
                <params id="IBM">IBM,5.0,5.0</params>
            </binding>
        </bindings>
</processor>
```

C.12 buffer-size

Use this element to define the size of the internal store buffer that's used to temporarily hold asynchronous updates that need to be written to the store. Does not support dynamic updates.

C.12.1 Child Elements

The buffer-size component configuration element has no child elements.

C.12.2 Attributes

The buffer-size component has no attributes.

C.12.3 Example

The following example shows how to use the buffer-size element in the component configuration file:

```
<caching-system>
    <name>providerCachingSystem</name>
    <cache>
        <name>providerCache</name>
        <max-size>1000</max-size>
        <eviction-policy>FIFO</eviction-policy>
        <time-to-live>60000</time-to-live>
        <idle-time>120000</idle-time>
        <write-behind>
            <work-manager-name>JettyWorkManager</work-manager-name>
            <batch-size>100</batch-size>
            <buffer-size>100</buffer-size>
            <buffer-write-attempts>100</buffer-write-attempts>
            <buffer-write-timeout>100</buffer-write-timeout>
        </write-behind>
        <work-manager-name>JettyWorkManager</work-manager-name>
```

```
            <listeners asynchronous="false">
                <work-manager-name>JettyWorkManager</work-manager-name>
            </listeners>
        </cache>
</caching-system>
```

C.13 buffer-write-attempts

Use this element to define the number of attempts that the user thread will make to write to the store buffer. The user thread is the thread that creates or updates a cache entry. If the user thread cannot write to the store buffer (all write attempts fail), it will invoke the store synchronously. This element may be changed dynamically.

C.13.1 Child Elements

The `buffer-write-attempts` component configuration element has no child elements.

C.13.2 Attributes

The `buffer-write-attempts` component has no attributes.

C.13.3 Example

The following example shows how to use the `buffer-write-attempts` element in the component configuration file:

```
<caching-system>
    <name>providerCachingSystem</name>
    <cache>
        <name>providerCache</name>
        <max-size>1000</max-size>
        <eviction-policy>FIFO</eviction-policy>
        <time-to-live>60000</time-to-live>
        <idle-time>120000</idle-time>
        <write-behind>
            <work-manager-name>JettyWorkManager</work-manager-name>
            <batch-size>100</batch-size>
            <buffer-size>100</buffer-size>
            <buffer-write-attempts>100</buffer-write-attempts>
            <buffer-write-timeout>100</buffer-write-timeout>
        </write-behind>
        <work-manager-name>JettyWorkManager</work-manager-name>
        <listeners asynchronous="false">
            <work-manager-name>JettyWorkManager</work-manager-name>
        </listeners>
    </cache>
</caching-system>
```

C.14 buffer-write-timeout

Use this element to define the time in milliseconds that the user thread will wait before aborting an attempt to write to the store buffer. The attempt to write to the store buffer fails only in case the buffer is full. After the timeout, further attempts may be made to write to the buffer based on the value of buffer-write-attempts. This element may be changed dynamically.

C.14.1 Child Elements

The `buffer-write-timeout` component configuration element has no child elements.

C.14.2 Attributes

The `buffer-write-timeout` component has no attributes.

C.14.3 Example

The following example shows how to use the `buffer-write-timeout` element in the component configuration file:

```
<caching-system>
    <name>providerCachingSystem</name>
    <cache>
        <name>providerCache</name>
        <max-size>1000</max-size>
        <eviction-policy>FIFO</eviction-policy>
        <time-to-live>60000</time-to-live>
        <idle-time>120000</idle-time>
        <write-behind>
            <work-manager-name>JettyWorkManager</work-manager-name>
            <batch-size>100</batch-size>
            <buffer-size>100</buffer-size>
            <buffer-write-attempts>100</buffer-write-attempts>
            <buffer-write-timeout>100</buffer-write-timeout>
        </write-behind>
        <work-manager-name>JettyWorkManager</work-manager-name>
        <listeners asynchronous="false">
            <work-manager-name>JettyWorkManager</work-manager-name>
        </listeners>
    </cache>
</caching-system>
```

C.15 cache

Use this element to define a cache for a component. A *cache* is a temporary storage area for events, created exclusively to improve the overall performance of your Oracle CEP application; it is not necessary for the application to function correctly. Oracle CEP applications can optionally publish or consume events to and from a cache to increase the availability of the events and increase the performance of their applications.

For more information, see Section 12.2, "Configuring an Oracle CEP Local Caching System and Cache".

C.15.1 Child Elements

The `cache` component configuration element supports the following child elements:

- name

- max-size

- eviction-policy

- time-to-live

- idle-time

- One of:

- – write-none

- – write-through

- – write-behind

- work-manager-name

- listeners

C.15.2 Attributes

The `cache` component has no attributes.

C.15.3 Example

The following example shows how to use the `cache` element in the component configuration file:

```
<caching-system>
    <name>providerCachingSystem</name>
    <cache>
        <name>providerCache</name>
        <max-size>1000</max-size>
        <eviction-policy>FIFO</eviction-policy>
        <time-to-live>60000</time-to-live>
        <idle-time>120000</idle-time>
        <write-none/>
        <work-manager-name>JettyWorkManager</work-manager-name>
        <listeners asynchronous="false">
            <work-manager-name>JettyWorkManager</work-manager-name>
        </listeners>
    </cache>
</caching-system>
```

C.16 caching-system

Use this element to define an Oracle CEP local caching system component. A *caching system* refers to a configured instance of a caching implementation. A caching system defines a named set of configured caches as well as the configuration for remote communication if any of the caches are distributed across multiple machines.

For more information, see Section 12.2.1, "Configuring an Oracle CEP Local Cache as an Event Listener".

C.16.1 Child Elements

The `caching-system` component configuration element supports the following child elements:

- name

- cache

C.16.2 Attributes

The `caching-system` component has no attributes.

C.16.3 Example

The following example shows how to use the `caching-system` element in the component configuration file:

```
<caching-system>
    <name>providerCachingSystem</name>
    <cache>
        <name>providerCache</name>
        <max-size>1000</max-size>
        <eviction-policy>FIFO</eviction-policy>
        <time-to-live>60000</time-to-live>
        <idle-time>120000</idle-time>
        <write-none/>
        <work-manager-name>JettyWorkManager</work-manager-name>
        <listeners asynchronous="false">
            <work-manager-name>JettyWorkManager</work-manager-name>
        </listeners>
    </cache>
</caching-system>
```

In the example, the channel's unique identifier is `providerCachingSystem`.

C.17 channel

Use this element to define a channel component. An Oracle CEP application contains one or more channel components that represent the physical conduit through which events flow between other types of components, such as between adapters and processors, and between processors and event beans (business logic POJOs).

C.17.1 Child Elements

The `channel` component configuration element supports the following child elements:

- name
- record-parameters
- playback-parameters
- max-size
- max-threads
- selector
- heartbeat-timeout

C.17.2 Attributes

The `channel` component has no attributes.

C.17.3 Example

The following example shows how to use the `channel` element in the component configuration file:

```
<channel>
    <name>MatchOutputChannel</name>
    <max-size>0</max-size>
    <max-threads>0</max-threads>
    <selector>match</selector>
</channel>
```

In the example, the channel's unique identifier is `MatchOutputChannel`.

C.18 channel (http-pub-sub-adapter Child Element)

Use the `channel` element to specify the channel that the http-pub-sub-adapter publishes or subscribes to, whichever is appropriate, for *all* http-pub-sub-adapter, whether they are local or remote or for publishing or subscribing.

C.18.1 Child Elements

The `channel` component configuration element has no child elements.

C.18.2 Attributes

The `channel` component has no attributes.

C.18.3 Example

The following example shows how to use the `channel` element in the component configuration file:

```
<http-pub-sub-adapter>
    <name>localPublisher</name>
    <server-context-path>/pubsub</server-context-path>
    <channel>/channel2</channel>
</http-pub-sub-adapter>
```

In the example, the `localPublisher` pub-sub adapter publishes to a local channel with pattern `/channel2`.

C.19 coherence-cache-config

Use this element to define the Oracle Coherence cache configuration for a coherence-caching-system.

For more information, see Section 12.2, "Configuring an Oracle CEP Local Caching System and Cache".

C.19.1 Child Elements

The `coherence-cache-config` component configuration element has no child elements.

C.19.2 Attributes

The `coherence-cache-config` component has no attributes.

C.19.3 Example

The following example shows how to use the `coherence-cache-config` element in the component configuration file:

```
<coherence-caching-system>
    <name>nativeCachingSystem</name>
    <coherence-cache-config>
        applications/cache_cql/coherence/coherence-cache-config.xml
    </coherence-cache-config>
```

```
    </coherence-caching-system>
```

C.20 coherence-caching-system

Use this element to define an Oracle Coherence caching system component. A *caching system* refers to a configured instance of a caching implementation. A caching system defines a named set of configured caches as well as the configuration for remote communication if any of the caches are distributed across multiple machines.

For more information, see Section 12.3, "Configuring an Oracle Coherence Caching System and Cache".

C.20.1 Child Elements

The `coherence-caching-system` component configuration element supports the following child elements:

- name
- coherence-cache-config
- coherence-cluster-config

C.20.2 Attributes

The `coherence-caching-system` component has no attributes.

C.20.3 Example

The following example shows how to use the `coherence-caching-system` element in the component configuration file:

```
<coherence-caching-system>
    <name>nativeCachingSystem</name>
    <coherence-cache-config>
        applications/cache_cql/coherence/coherence-cache-config.xml
    </coherence-cache-config>
</coherence-caching-system>
```

In the example, the channel's unique identifier is `nativeCachingSystem`.

C.21 coherence-cluster-config

Use this element to define the Oracle Coherence cluster configuration for a coherence-caching-system.

For more information, see "Overview of Oracle CEP Multi-Server Domain Administration" in the *Oracle CEP Administrator's Guide*.

C.21.1 Child Elements

The `coherence-cache-config` component configuration element has no child elements.

C.21.2 Attributes

The `coherence-cache-config` component has no attributes.

C.21.3 Example

The following example shows how to use the `coherence-cache-config` element in the component configuration file:

```
<coherence-caching-system>
    <name>nativeCachingSystem</name>
    <coherence-cluster-config>
        applications/cluster_cql/coherence/coherence-cluster-config.xml
    </coherence-cluster-config>
</coherence-caching-system>
```

C.22 collect-interval

Use this element to define the collection interval of an average-latency or max-latency element in a diagnostic profile.

For more information, see "Monitoring the Throughput and Latency of a Stage or Path in the EPN" in the *Oracle CEP Visualizer User's Guide*.

C.22.1 Child Elements

The `collect-interval` component configuration element supports the following child elements:

- amount

- unit

C.22.2 Attributes

The `collect-interval` component has no attributes.

C.22.3 Example

The following example shows how to use the `collect-interval` element in the component configuration file:

```
<diagnostic-profiles>
    <name>myselfprofiles</name>
    <profile>
        <name>testProfile0</name>
        <enabled>true</enabled>
        <start-stage>MetricSubscriber</start-stage>
        <max-latency>
            <collect-interval>
                <amount>1000</amount>
                <unit>s</unit>
            </collect-interval>
            <name>testProfile0MaxLat</name>
            <start-location>
                <application>diagnostic</application>
                <stage>MetricSubscriber</stage>
                <direction>INBOUND</direction>
            </start-location>
            <end-location>
                <application>diagnostic</application>
                <stage>MonitorProcessor</stage>
                <direction>OUTBOUND</direction>
            </end-location>
        </max-latency>
    </profile>
```

```
</diagnostic-profiles>
```

C.23 concurrent-consumers

Use this element to define the number of consumers to create. Default value is 1. If you set this value to number greater than one, be sure that your converter bean is thread-safe. This is because the converter bean will be shared among the consumers.

C.23.1 Child Elements

The `concurrent-consumers` component configuration element has no child elements.

C.23.2 Attributes

The `concurrent-consumers` component has no attributes.

C.23.3 Example

The following example shows how to use the `concurrent-consumers` element in the component configuration file:

```
<jms-adapter>
    <name>jmsInbound-text</name>
    <connection-jndi-name>weblogic.jms.ConnectionFactory</connection-jndi-name>
    <destination-name>JMSServer-0/Module1!Queue1</destination-name>
    <user>weblogic</user>
    <password>weblogic</password>
    <work-manager>JettyWorkManager</work-manager>
    <concurrent-consumers>1</concurrent-consumers>
    <session-transacted>false</session-transacted>
</jms-adapter>
```

C.24 connection-jndi-name

Use this optional element to define a JNDI name of the JMS connection factory. Default value is `weblogic.jms.ConnectionFactory` for Oracle CEP server JMS.

C.24.1 Child Elements

The `connection-jndi-name` component configuration element has no child elements.

C.24.2 Attributes

The `connection-jndi-name` component has no attributes.

C.24.3 Example

The following example shows how to use the `connection-jndi-name` element in the component configuration file:

```
<jms-adapter>
    <name>jmsInbound-text</name>
    <connection-jndi-name>weblogic.jms.ConnectionFactory</connection-jndi-name>
    <destination-name>JMSServer-0/Module1!Queue1</destination-name>
    <user>weblogic</user>
    <password>weblogic</password>
    <work-manager>JettyWorkManager</work-manager>
```

```
        <concurrent-consumers>1</concurrent-consumers>
        <session-transacted>false</session-transacted>
</jms-adapter>
```

C.25 database

Use this element to define a database reference for an EPL processor component.

For more information, see Chapter 11, "Configuring EPL Processors".

C.25.1 Child Elements

The database component configuration element has no child elements.

C.25.2 Attributes

Table C–2 lists the attributes of the database component configuration element.

Table C–2   Attributes of the database Component Configuration Element

Attribute	Description	Data Type	Required?
name	Unique identifier for this query.	String	Yes.
data-source-name	The name of the data source as defined in the Oracle CEP server config.xml file.	String	Yes.

C.25.3 Example

The following example shows how to use the database element in the component configuration file:

```
<processor>
    <name>proc</name>
    <rules>
        <rule id="rule1"><![CDATA[
            SELECT symbol, price
            FROM ExchangeEvent retain 1 event,
            StockDb ('SELECT symbol FROM Stock WHERE symbol = ${symbol}')
        ]]></rule>
    </rules>

    <database name="StockDb" data-source-name="StockDs" />

</processor>
```

C.26 dataset-name

Use this element to define the group of data that the user wants to group together. In the case of the Oracle RDBMS-based provider, it specifies the database area, or schema, in which the tables that store the recorded events are created. When configuring the Oracle RDBMS-based provider, you are required to specify this element.

C.26.1 Child Elements

The dataset-name component configuration element has no child elements.

C.26.2 Attributes

The dataset-name component has no attributes.

C.26.3 Example

The following example shows how to use the dataset-name element in the component configuration file:

```
<record-parameters>
    <dataset-name>tuple1</dataset-name>
        <event-type-list>
            <event-type>TupleEvent1</event-type>
        </event-type-list>
        <provider-name>test-rdbms-provider</provider-name>
    <batch-size>1</batch-size>
    <batch-time-out>10</batch-time-out>
</record-parameters>
```

C.27 delivery-mode

Use this element to define the delivery mode for a jms-adapter.

Valid values are:

- persistent (default)

- nonpersistent

C.27.1 Child Elements

The delivery-mode component configuration element has no child elements.

C.27.2 Attributes

The delivery-mode component has no attributes.

C.27.3 Example

The following example shows how to use the delivery-mode element in the component configuration file:

```
<jms-adapter>
    <name>jmsOutbound-map</name>
    <event-type>JMSTestEvent</event-type>
    <jndi-provider-url>t3://localhost:7001</jndi-provider-url>
    <destination-jndi-name>Topic1</destination-jndi-name>
    <delivery-mode>nonpersistent</delivery-mode>
</jms-adapter>
```

C.28 destination-jndi-name

Use this required element to define the JMS destination name for a jms-adapter.

Specify either the JNDI name or the actual destination-name, but not both.

C.28.1 Child Elements

The destination-jndi-name component configuration element has no child elements.

C.28.2 Attributes

The destination-jndi-name component has no attributes.

C.28.3 Example

The following example shows how to use the `destination-jndi-name` element in the component configuration file:

```
<jms-adapter>
    <name>jmsOutbound-map</name>
    <event-type>JMSTestEvent</event-type>
    <jndi-provider-url>t3://localhost:7001</jndi-provider-url>
    <destination-jndi-name>Topic1</destination-jndi-name>
    <delivery-mode>nonpersistent</delivery-mode>
</jms-adapter>
```

C.29 destination-name

Use this required element to define the JMS destination name for a jms-adapter.

Specify either the actual destination name or the destination-jndi-name, but not both.

C.29.1 Child Elements

The `destination-name` component configuration element has no child elements.

C.29.2 Attributes

The `destination-name` component has no attributes.

C.29.3 Example

The following example shows how to use the `destination-name` element in the component configuration file:

```
<jms-adapter>
    <name>jmsInbound-text</name>
    <connection-jndi-name>weblogic.jms.ConnectionFactory</connection-jndi-name>
    <destination-name>JMSServer-0/Module1!Queue1</destination-name>
    <user>weblogic</user>
    <password>weblogic</password>
    <work-manager>JettyWorkManager</work-manager>
    <concurrent-consumers>1</concurrent-consumers>
    <session-transacted>false</session-transacted>
</jms-adapter>
```

C.30 diagnostic-profiles

Use this element to define one or more Oracle CEP diagnostic profiles.

For more information, see "Monitoring the Throughput and Latency of a Stage or Path in the EPN" in the *Oracle CEP Visualizer User's Guide*.

C.30.1 Child Elements

The `diagnostics-profiles` component configuration element supports the following child elements:

- name
- profile

C.30.2 Attributes

The `diagnostics-profiles` component has no attributes.

C.30.3 Example

The following example shows how to use the `diagnostics-profiles` element in the component configuration file:

```
<diagnostics-profiles>
    <name>myDiagnosticProfiles</name>
    <profile>
        ...
    </profile>
</diagnostics-profiles>
```

In the example, the channel's unique identifier is `myDiagnosticProfiles`.

C.31 direction

Use this element to define the direction for a diagnostic profile end-location or start-location.

Valid values are:

- INBOUND

- OUTBOUND

For more information, see "Monitoring the Throughput and Latency of a Stage or Path in the EPN" in the *Oracle CEP Visualizer User's Guide*.

C.31.1 Child Elements

The `direction` component configuration has no child elements:

C.31.2 Attributes

The `direction` component has no attributes.

C.31.3 Example

The following example shows how to use the `direction` element in the component configuration file:

```
<diagnostic-profiles>
    <name>myselfprofiles</name>
    <profile>
        <name>testProfile0</name>
        <enabled>true</enabled>
        <start-stage>MetricSubscriber</start-stage>
        <max-latency>
            <collect-interval>
                <amount>1000</amount>
                <unit>s</unit>
            </collect-interval>
            <name>testProfile0MaxLat</name>
            <start-location>
                <application>diagnostic</application>
                <stage>MetricSubscriber</stage>
                <direction>INBOUND</direction>
            </start-location>
            <end-location>
```

```
            <application>diagnostic</application>
            <stage>MonitorProcessor</stage>
            <direction>OUTBOUND</direction>
         </end-location>
      </max-latency>
   </profile>
</diagnostic-profiles>
```

C.32 duration

Use this element to define a time duration for a schedule-time-range-offset or time-range-offset element in the form:

```
HH:MM:SS
```

Where: HH is a number of hours, MM is a number of minutes, and SS is a number of seconds.

C.32.1 Child Elements

The duration component configuration element has no child elements.

C.32.2 Attributes

The duration component has no attributes.

C.32.3 Example

The following example shows how to use the duration element in the component configuration file:

```
<record-parameters>
    <dataset-name>tuple1</dataset-name>
    <event-type-list>
        <event-type>TupleEvent1</event-type>
    </event-type-list>
    <provider-name>test-rdbms-provider</provider-name>
    <store-policy-parameters>
        <parameter>
            <name>timeout</name>
            <value>300</value>
        <parameter>
    </store-policy-parameters>
    <time-range-offset>
        <start>04-07-2009:06:00:00</start>
        <duration>03:00:00</duration>
    </time-range-offset>
    <batch-size>1</batch-size>
    <batch-time-out>10</batch-time-out>
</record-parameters>
```

C.33 enabled

Use this element to define whether or not a diagnostic profile is enabled.

Valid values are:

- true
- false

For more information, see "Monitoring the Throughput and Latency of a Stage or Path in the EPN" in the *Oracle CEP Visualizer User's Guide*.

C.33.1 Child Elements

The `enabled` component configuration element has no child elements.

C.33.2 Attributes

The `enabled` component has no attributes.

C.33.3 Example

The following example shows how to use the `enabled` element in the component configuration file:

```
<diagnostic-profiles>
    <name>myselfprofiles</name>
    <profile>
        <name>testProfile0</name>
        <enabled>true</enabled>
        <start-stage>MetricSubscriber</start-stage>
        <max-latency>
            <start-location>
                <application>diagnostic</application>
                <stage>MetricSubscriber</stage>
                <direction>INBOUND</direction>
            </start-location>
            <end-location>
                <application>diagnostic</application>
                <stage>MonitorProcessor</stage>
                <direction>OUTBOUND</direction>
            </end-location>
        </max-latency>
    </profile>
</diagnostic-profiles>
```

C.34 encrypted-password

Use the `encrypted-password` element in the following parent elements:

- http-pub-sub-adapter: Use the `encrypted-password` element to define the encrypted password if the HTTP pub-sub server to which the Oracle CEP application is publishing requires user authentication.

- jms-adapter: Use the `encrypted-password` element to define the encrypted password if the JMS service provider requires user authentication.

Use either `encrypted-password` or password but not both.

C.34.1 Child Elements

The `encrypted-password` component configuration element has no child elements.

C.34.2 Attributes

The `encrypted-password` component has no attributes.

C.34.3 Example

The following example shows how to use the `encrypted-password` element in the component configuration file:

```
<http-pub-sub-adapter>
    <name>remotePub</name>
    <server-url>http://localhost:9002/pubsub</server-url>
    <channel>/test1</channel>
    <event-type>com.bea.wlevs.tests.httppubsub.PubsubTestEvent</event-type>
    <user>wlevs</user>
    <encrypted-password>{Salted-3DES}s4YUEvH4Wl2DAjb45iJnrw==</encrypted-password>
</http-pub-sub-adapter>
```

C.35 end

Use this element to define an end time for a time-range element using the following format:

```
MM-DD-YYYY:HH:MM:SS
```

Where `MM` is the two-digit month, `DD` is the two-digit day of the month, `YYYY` is the four-digit year, `HH` is the hour (in 24-hour format), `MM` is the two-digit minutes, and `SS` is the two-digit seconds.

C.35.1 Child Elements

The `end` component configuration element has no child elements.

C.35.2 Attributes

The `end` component has no attributes.

C.35.3 Example

The following example shows how to use the `end` element in the component configuration file:

```
<record-parameters>
    <dataset-name>tuple1</dataset-name>
    <event-type-list>
        <event-type>TupleEvent1</event-type>
    </event-type-list>
    <provider-name>test-rdbms-provider</provider-name>
    <store-policy-parameters>
        <parameter>
            <name>timeout</name>
            <value>300</value>
        <parameter>
    </store-policy-parameters>
    <time-range>
        <start>04-07-2009:06:00:00</start>
        <end>04-10-2009:22:00:00</end>
    </time-range>
    <batch-size>1</batch-size>
    <batch-time-out>10</batch-time-out>
</record-parameters>
```

C.36 end-location

Use this element to define the end location of a average-latency or max-latency element in a diagnostic profile.

For more information, see "Monitoring the Throughput and Latency of a Stage or Path in the EPN" in the *Oracle CEP Visualizer User's Guide*.

C.36.1 Child Elements

The `end-location` component configuration element supports the following child elements:

- application

- stage

- direction

C.36.2 Attributes

The `end-location` component has no attributes.

C.36.3 Example

The following example shows how to use the `end-location` element in the component configuration file:

```
<diagnostic-profiles>
    <name>myselfprofiles</name>
    <profile>
        <name>testProfile0</name>
        <enabled>true</enabled>
        <start-stage>MetricSubscriber</start-stage>
        <max-latency>
            <collect-interval>
                <amount>1000</amount>
                <unit>s</unit>
            </collect-interval>
            <name>testProfile0MaxLat</name>
            <start-location>
                <application>diagnostic</application>
                <stage>MetricSubscriber</stage>
                <direction>INBOUND</direction>
            </start-location>
            <end-location>
                <application>diagnostic</application>
                <stage>MonitorProcessor</stage>
                <direction>OUTBOUND</direction>
            </end-location>
        </max-latency>
    </profile>
</diagnostic-profiles>
```

C.37 event-bean

Use this element to define an event bean component.

For more information, see Chapter 8, "Configuring Custom Adapters, Event Beans, and Spring Beans".

C.37.1 Child Elements

The `event-bean` component configuration element supports the following child elements:

- name
- record-parameters
- playback-parameters

C.37.2 Attributes

The `event-bean` component has no attributes.

C.37.3 Example

The following example shows how to use the `event-bean` element in the component configuration file:

```
<event-bean>
    <name>myEventBean</name>
</event-bean>
```

In the example, the channel's unique identifier is `myEventBean`.

C.38 event-type

Use the `event-type` element in the following parent elements:

- http-pub-sub-adapter: Use the `event-type` element to specify the JavaBean to which incoming messages are mapped for an http-pub-sub-adapter you configured for subscribing. This setting is mandatory.

  You can also optionally use the `event-type` element in a pub-sub adapter for publishing if you want to limit the types of events that are published to just those specified by the `event-type` elements. Otherwise, all events sent to the pub-sub adapter are published. For example:

  At runtime, Oracle CEP uses the incoming key-value pairs in the message to map the message data to the specified event type.

- jms-adapter: Use the `event-type` element to specify an event type whose properties match the JMS message properties. Specify this child element only if you want Oracle CEP to automatically perform the conversion between JMS messages and events. If you have created your own custom converter bean, then do not specify this element.

- record-parameters: Use the `event-type` element to specify an event that you want to record.

The value of the `event-type` element must be one of the event types you defined in your event type repository. For more information, see Section 1.1.2, "Event Types".

C.38.1 Child Elements

The `event-type` component configuration element has no child elements.

C.38.2 Attributes

The `event-type` component has no attributes.

C.38.3 Example

The following example shows how to use the event-type element in the component configuration file:

```
<record-parameters>
    <dataset-name>tuple1</dataset-name>
        <event-type-list>
            <event-type>TupleEvent1</event-type>
        </event-type-list>
        <provider-name>test-rdbms-provider</provider-name>
    <batch-size>1</batch-size>
    <batch-time-out>10</batch-time-out>
</record-parameters>
```

C.39 event-type-list

Use this element to define one or more events for record or playback for a component.

C.39.1 Child Elements

The event-type-list component configuration element supports the following child elements:

■ event-type

C.39.2 Attributes

The event-type-list component has no attributes.

C.39.3 Example

The following example shows how to use the event-type-list element in the component configuration file:

```
<record-parameters>
    <dataset-name>tuple1</dataset-name>
        <event-type-list>
            <event-type>TupleEvent1</event-type>
        </event-type-list>
        <provider-name>test-rdbms-provider</provider-name>
    <batch-size>1</batch-size>
    <batch-time-out>10</batch-time-out>
</record-parameters>
```

C.40 eviction-policy

Use this element to define the eviction policy the cache uses when max-size is reached.

Valid values are:

■ FIFO: first in, first out.

■ LRU: least recently used

■ LFU: least frequently used (default)

■ NRU: not recently used

C.40.1 Child Elements

The `eviction-policy` component configuration element has no child elements.

C.40.2 Attributes

The `eviction-policy` component has no attributes.

C.40.3 Example

The following example shows how to use the `eviction-policy` element in the component configuration file:

```
<caching-system>
    <name>providerCachingSystem</name>
    <cache>
        <name>providerCache</name>
        <max-size>1000</max-size>
        <eviction-policy>FIFO</eviction-policy>
        <time-to-live>60000</time-to-live>
        <idle-time>120000</idle-time>
        <write-none/>
        <work-manager-name>JettyWorkManager</work-manager-name>
        <listeners asynchronous="false">
            <work-manager-name>JettyWorkManager</work-manager-name>
        </listeners>
    </cache>
</caching-system>
```

C.41 heartbeat-timeout

Use this element to define a heartbeat timeout for a system-timestamped `channel` component.

For system-timestamped relations or streams, time is dependent upon the arrival of data on the relation or stream data source. Oracle CEP generates a heartbeat on a system timestamped relation or stream if there is no activity (no data arriving on the stream or relation's source) for more than this number of nanoseconds. Either the relation or stream is populated by its specified source or Oracle CEP generates a heartbeat every `heartbeat-timeout` number of nanoseconds.

The heartbeat child element applies to system-timestamped relations or streams only when no events arrive in the event channels that are feeding the processors and the processor has been configured with a statement that includes some temporal operator, such as a time-based window or a pattern matching with duration.

> **Note:** This attribute is only applicable when a non-streaming source is connected to the channel

C.41.1 Child Elements

The `heartbeat-timeout` component configuration element has no child elements.

C.41.2 Attributes

The `heartbeat-timeout` component configuration element has no attributes.

C.41.3 Example

The following example shows how to use the `channel` element in the component configuration file:

```
<channel>
    <name>MatchOutputChannel</name>
    <max-size>0</max-size>
    <max-threads>0</max-threads>
    <selector>match</selector>
    <heartbeat-timeout>10000</heartbeat-timeout>
</channel>
```

In the example, the channel's unique identifier is `MatchOutputChannel`.

C.42 http-pub-sub-adapter

Use this element to define an HTTP publish-subscribe server adapter component.

For more information, see Chapter 7, "Configuring HTTP Publish-Subscribe Server Adapters".

C.42.1 Child Elements

The `http-pub-sub-adapter` component configuration element supports the following child elements:

- name
- record-parameters
- playback-parameters
- symbols
- work-manager-name
- netio
- One of:
  - server-context-path
  - server-url
- event-type
- user
- One of:
  - password
  - encrypted-password

C.42.2 Attributes

The `http-pub-sub-adapter` component configuration element has no attributes.

C.42.3 Example

The following example shows how to use the `http-pub-sub-adapter` element in the component configuration file:

```
<http-pub-sub-adapter>
    <name>remotePub</name>
```

```
        <server-url>http://localhost:9002/pubsub</server-url>
        <channel>/test1</channel>
        <event-type>com.bea.wlevs.tests.httppubsub.PubsubTestEvent</event-type>
        <user>wlevs</user>
        <password>wlevs</password>
</http-pub-sub-adapter>
```

In the example, the adapter's unique identifier is `remotePub`.

C.43 idle-time

Use this element to define the number of milliseconds a cache entry may not be accessed before being actively removed from the cache. By default, there is no idle-time set. This element may be changed dynamically.

C.43.1 Child Elements

The `idle-time` component configuration element has no child elements.

C.43.2 Attributes

The `idle-time` component has no attributes.

C.43.3 Example

The following example shows how to use the `idle-time` element in the component configuration file:

```
<caching-system>
    <name>providerCachingSystem</name>
    <cache>
        <name>providerCache</name>
        <max-size>1000</max-size>
        <eviction-policy>FIFO</eviction-policy>
        <time-to-live>60000</time-to-live>
        <idle-time>120000</idle-time>
        <write-none/>
        <work-manager-name>JettyWorkManager</work-manager-name>
        <listeners asynchronous="false">
            <work-manager-name>JettyWorkManager</work-manager-name>
        </listeners>
    </cache>
</caching-system>
```

C.44 jms-adapter

Use this element to define a JMS adapter component.

For more information, see Chapter 6, "Configuring JMS Adapters".

C.44.1 Child Elements

The `jms-adapter` component configuration element supports the following child elements:

- name
- record-parameters
- playback-parameters

- event-type

- jndi-provider-url

- connection-jndi-name

- One of:

    - destination-jndi-name

    - destination-name

- user

    One of:

    - password

    - encrypted-password

- work-manager

- concurrent-consumers

- message-selector

- session-ack-mode-name

- session-transacted

- delivery-mode

C.44.2 Attributes

The `jms-adapter` component configuration element has no attributes.

C.44.3 Example

The following example shows how to use the `jms-adapter` element in the component configuration file:

```
<jms-adapter>
    <name>jmsInbound-text</name>
    <jndi-provider-url>t3://localhost:7001</jndi-provider-url>
    <destination-name>JMSServer-0/Module1!Queue1</destination-name>
    <user>weblogic</user>
    <password>weblogic</password>
    <work-manager>JettyWorkManager</work-manager>
    <concurrent-consumers>1</concurrent-consumers>
    <session-transacted>false</session-transacted>
</jms-adapter>
```

In the example, the adapter's unique identifier is `jmsInbound-text`.

C.45 jndi-factory

Use this optional element to define a JNDI factory for a jms-adapter. The JNDI factory name. Default value is `weblogic.jndi.WLInitialContextFactory`, for Oracle CEP server JMS

C.45.1 Child Elements

The `jndi-factory` component configuration element has no child elements.

C.45.2 Attributes

The `jndi-factory` component has no attributes.

C.45.3 Example

The following example shows how to use the `jndi-provider-url` element in the component configuration file:

```
<jms-adapter>
    <name>jmsInbound-text</name>
    <jndi-factory>weblogic.jndi.WLInitialContextFactory</jndi-factory>
    <destination-name>JMSServer-0/Module1!Queue1</destination-name>
    <user>weblogic</user>
    <password>weblogic</password>
    <work-manager>JettyWorkManager</work-manager>
    <concurrent-consumers>1</concurrent-consumers>
    <session-transacted>false</session-transacted>
</jms-adapter>
```

C.46 jndi-provider-url

Use this required element to define a JNDI provider URL for a jms-adapter.

C.46.1 Child Elements

The `jndi-provider-url` component configuration element has no child elements.

C.46.2 Attributes

The `jndi-provider-url` component has no attributes.

C.46.3 Example

The following example shows how to use the `jndi-provider-url` element in the component configuration file:

```
<jms-adapter>
    <name>jmsInbound-text</name>
    <jndi-provider-url>t3://localhost:7001</jndi-provider-url>
    <destination-name>JMSServer-0/Module1!Queue1</destination-name>
    <user>weblogic</user>
    <password>weblogic</password>
    <work-manager>JettyWorkManager</work-manager>
    <concurrent-consumers>1</concurrent-consumers>
    <session-transacted>false</session-transacted>
</jms-adapter>
```

C.47 listeners

Use this element to define the behavior for cache listeners.

The `listeners` element has a single child element, work-manager-name, that specifies the work manager to be used for asynchronously invoking listeners. This value is ignored if synchronous invocations are enabled. If a work manager is specified for the cache itself, this value overrides it for invoking listeners only.

C.47.1 Child Elements

The listeners component configuration element supports the following child elements:

- work-manager-name

C.47.2 Attributes

Table C–3 lists the attributes of the listeners component configuration element.

Table C–3   Attributes of the listeners Component Configuration Element

Attribute	Description	Data Type	Required?
asynchronous	Execute listeners asynchronously. Valid values are true and false. Default value is false.	Boolean	No.

C.47.3 Example

The following example shows how to use the listeners element in the component configuration file:

```
<caching-system>
    <name>providerCachingSystem</name>
    <cache>
        <name>providerCache</name>
        <max-size>1000</max-size>
        <eviction-policy>FIFO</eviction-policy>
        <time-to-live>60000</time-to-live>
        <idle-time>120000</idle-time>
        <write-behind>
            <work-manager-name>JettyWorkManager</work-manager-name>
            <batch-size>100</batch-size>
            <buffer-size>100</buffer-size>
            <buffer-write-attempts>100</buffer-write-attempts>
            <buffer-write-timeout>100</buffer-write-timeout>
        </write-behind>
        <work-manager-name>JettyWorkManager</work-manager-name>
        <listeners asynchronous="false">
            <work-manager-name>JettyWorkManager</work-manager-name>
        </listeners>
    </cache>
</caching-system>
```

C.48 location

Use this element to define the location of a throughput element in a diagnostic profile.

For more information, see "Monitoring the Throughput and Latency of a Stage or Path in the EPN" in the *Oracle CEP Visualizer User's Guide*.

C.48.1 Child Elements

The location component configuration element supports the following child elements:

- application
- stage
- direction

C.48.2 Attributes

The location component has no attributes.

C.48.3 Example

The following example shows how to use the location element in the component
configuration file:

```
<diagnostic-profiles>
    <name>myselfprofiles</name>
    <profile>
        <name>testProfile0</name>
        <enabled>true</enabled>
        <start-stage>MetricSubscriber</start-stage>
        <throughput>
            <throughput-interval>
                <amount>100000</amount>
                <unit>MICROSECONDS</unit>
            </throughput-interval>
            <average-interval>
                <amount>100000000</amount>
                <unit>NANOSECONDS</unit>
            </average-interval>
            <location>
                <application>diagnostic</application>
                <stage>AlertEventStream</stage>
                <direction>INBOUND</direction>
            </location>
        </throughput>
    </profile>
</diagnostic-profiles>
```

C.49 max-latency

Use this element to define the maximum latency calculation of a diagnostic profile.

For more information, see "Monitoring the Throughput and Latency of a Stage or Path
in the EPN" in the *Oracle CEP Visualizer User's Guide*.

C.49.1 Child Elements

The max-latency component configuration element supports the following child
elements:

- name
- collect-interval
- start-location
- end-location

C.49.2 Attributes

The max-latency component has no attributes.

C.49.3 Example

The following example shows how to use the max-latency element in the
component configuration file:

```
<diagnostic-profiles>
```

```
                    <name>myselfprofiles</name>
                    <profile>
                        <name>testProfile0</name>
                        <enabled>true</enabled>
                        <start-stage>MetricSubscriber</start-stage>
                        <max-latency>
                            <start-location>
                                <application>diagnostic</application>
                                <stage>MetricSubscriber</stage>
                                <direction>INBOUND</direction>
                            </start-location>
                            <end-location>
                                <application>diagnostic</application>
                                <stage>MonitorProcessor</stage>
                                <direction>OUTBOUND</direction>
                            </end-location>
                        </max-latency>
                    </profile>
            </diagnostic-profiles>
```

C.50 max-size

Use the max-size element in the following parent elements:

- channel or stream: Use the max-size child element to specify the maximum size of the channel. Zero-size channels synchronously pass-through events. Non-zero size channels process events asynchronously, buffering events by the requested size. If max-threads is zero, then max-size is zero. The default value is 0.

- cache: Use the max-size element to define the number of cache elements in memory after which eviction or paging occurs. Currently, the maximum cache size is 2^31-1 entries. This element may be changed dynamically

C.50.1 Child Elements

The max-size component configuration element has no child elements.

C.50.2 Attributes

The max-size component has no attributes.

C.50.3 Example

The following example shows how to use the max-size element in the component configuration file:

```
<stream>
    <name>monitoring-control-stream</name>
    <max-size>10000</max-size>
    <max-threads>1</max-threads>
</stream>
```

C.51 max-threads

Use this element to define the maximum number of threads that Oracle CEP server uses to process events for a channel or stream. The default value is 0..

When set to 0, the channel acts as a pass-through. When max-threads > 0, the channel acts as classic blocking queue, where upstream components are producers of

events and the downstream components are the consumers of events. The queue size is defined by the configuration max-size. There will be up to max-threads number of threads consuming events from the queue.

You can change max-threads from 0 to a positive integer (that is, from a pass through to multiple threads) without redeploying. However, if you change max-threads from a positive integer to 0 (that is, from multiple threads to a pass through), then you must redeploy your application.

If the max-size attribute is 0, then setting a value for max-threads has no effect.

The default value for this attribute is 0.

Setting this value has no effect when max-size is 0.

C.51.1 Child Elements

The max-threads component configuration element has no child elements.

C.51.2 Attributes

The max-threads component has no attributes.

C.51.3 Example

The following example shows how to use the max-threads element in the component configuration file:

```
<channel>
    <name>monitoring-control-stream</name>
    <max-size>10000</max-size>
    <max-threads>1</max-threads>
</channel>
```

C.52 message-selector

Use this element to JMS message selector to use to filter messages in a jms-adapter.

The syntax of a message selector expression is based on a subset of the SQL92 conditional expression syntax and message headers and properties. For example, to select messages based on property EventType, you could use:

```
EventType = 'News' OR 'Commentary'
```

C.52.1 Child Elements

The message-selector component configuration element has no child elements.

C.52.2 Attributes

The message-selector component has no attributes.

C.52.3 Example

The following example shows how to use the message-selector element in the component configuration file:

```
<jms-adapter>
    <name>jmsInbound-text</name>
```

```
                  <connection-jndi-name>weblogic.jms.ConnectionFactory</connection-jndi-name>
                  <destination-name>JMSServer-0/Module1!Queue1</destination-name>
                  <user>weblogic</user>
                  <password>weblogic</password>
                  <work-manager>JettyWorkManager</work-manager>
                  <message-selector>EventType = 'News' OR 'Commentary'</message-selector>
                  <session-transacted>false</session-transacted>
          </jms-adapter>
```

C.53 name

Use the `name` element in the following parent elements:

- adapter, http-pub-sub-adapter, jms-adapter, processor (EPL), processor (Oracle CQL), stream, channel, event-bean, caching-system, and coherence-caching-system: Use the `name` element to associate this application configuration element with its corresponding element in the EPN assembly file. Valid value is the corresponding EPN assembly file element `id` attribute.

- diagnostic-profiles: Use the `name` element to uniquely identify the `diagnostic-profiles` element and each of its `profile` child elements.

- parameter: Use the `name` element to define the name of a name/value pair.

C.53.1 Child Elements

The `name` component configuration element has no child elements:

C.53.2 Attributes

The `name` component has no attributes.

C.53.3 Example

The following example shows how to use the `name` element in the component configuration file:

```
<diagnostics-profiles>
    <name>myDiagnosticProfiles</name>
    <profile>
        ...
    </profile>
</diagnostics-profiles>
```

In the example, the channel's unique identifier is `myDiagnosticProfiles`.

C.54 netio

Use this element to define a network input/output port for a component.

C.54.1 Child Elements

The `netio` component configuration element supports the following child elements:

- provider-name
- num-threads
- accept-backlog

C.54.2 Attributes

The `netio` component has no attributes.

C.54.3 Example

The following example shows how to use the `netio` element in the component configuration file:

```
<netio>
        <provider-name>providerCache</provider-name>
        <num-threads>1000</num-threads>
</netio>
```

C.55 num-threads

Use this element to define the number of threads in a network input/output port for a component.

C.55.1 Child Elements

The `num-threads` component configuration element has no child elements.

C.55.2 Attributes

The `num-threads` component has no attributes.

C.55.3 Example

The following example shows how to use the `num-threads` element in the component configuration file:

```
<netio>
        <provider-name>providerCache</provider-name>
        <num-threads>1000</num-threads>
</netio>
```

C.56 parameter

Use this element to define a name/value parameter for a component.

C.56.1 Child Elements

The `parameter` component configuration element supports the following child elements:

- name

- value

C.56.2 Attributes

The `parameter` component has no attributes.

C.56.3 Example

The following example shows how to use the `parameter` element in the component configuration file:

```
<record-parameters>
    <dataset-name>tuple1</dataset-name>
        <event-type-list>
            <event-type>TupleEvent1</event-type>
        </event-type-list>
        <provider-name>test-rdbms-provider</provider-name>
        <store-policy-parameters>
            <parameter>
                <name>timeout</name>
                <value>300</value>
            <parameter>
        </store-policy-parameters>
    <batch-size>1</batch-size>
    <batch-time-out>10</batch-time-out>
</record-parameters>
```

C.57 params

Use this element to define the parameters for a binding element.

The value of this element is a comma separated list of simple type values. The order of the values must correspond with the order of the parameters in the EPL rule associated with this `binding`.

For more information, see "Parameterized Queries" in the *Oracle CEP EPL Language Reference*.

C.57.1 Child Elements

The `params` component configuration element has no child elements.

C.57.2 Attributes

Table C–4 lists the attributes of the `params` component configuration element.

Table C–4   Attributes of the params Component Configuration Element

Attribute	Description	Data Type	Required?
`id`	Unique identifier for this `params` element.	String	No.

C.57.3 Example

The following example shows how to use the `params` element in the component configuration file:

```
<processor>
    <name>processor1</name>
        <record-parameters>
            <dataset-name>test1data</dataset-name>
            <event-type-list>
                <event-type>SimpleEvent</event-type>
            </event-type-list>
            <provider-name>test-rdbms-provider</provider-name>
            <batch-size>1</batch-size>
            <batch-time-out>10</batch-time-out>
        </record-parameters>
        <rules>
```

```
            <rule id="rule1"><![CDATA[
                select stockSymbol, avg(price) as percentage
                from StockTick retain 5 events
                where stockSymbol=?
                having avg(price)  > ? or avg(price)  < ?
            ]]></rule>
        </rules>
        <bindings>
            <binding id="rule1">
                <params>BEAS,10.0,-10.0</params>
                <params id="IBM">IBM,5.0,5.0</params>
            </binding>
        </bindings>
</processor>
```

C.58 password

Use the `password` element in the following parent elements:

- http-pub-sub-adapter: Use the `password` element to define the user password if the HTTP pub-sub server to which the Oracle CEP application is publishing requires user authentication.

- jms-adapter: Use the `password` element to define the user password if the JMS service provider requires user authentication.

Use either encrypted-password or `password` but not both.

C.58.1 Child Elements

The `password` component configuration element has no child elements.

C.58.2 Attributes

The `password` component has no attributes.

C.58.3 Example

The following example shows how to use the `password` element in the component configuration file:

```
<http-pub-sub-adapter>
    <name>remotePub</name>
    <server-url>http://localhost:9002/pubsub</server-url>
    <channel>/test1</channel>
    <event-type>com.bea.wlevs.tests.httppubsub.PubsubTestEvent</event-type>
    <user>wlevs</user>
    <password>wlevs</password>
</http-pub-sub-adapter>
```

C.59 playback-parameters

Use this element to define event playback parameters for a component.

For more information, see Chapter 13, "Configuring Event Record and Playback".

C.59.1 Child Elements

The `playback-parameters` component configuration element supports the following child elements:

- dataset-name

- event-type-list

- provider-name

- store-policy-parameters

- max-size

- max-threads

- One of:

  – time-range

  – time-range-offset

- One of:

  – schedule-time-range

  – schedule-time-range-offset

- playback-speed

- repeat

C.59.2 Attributes

The `playback-parameters` component has no attributes.

C.59.3 Example

The following example shows how to use the `playback-parameters` element in the component configuration file:

```
<playback-parameters>
    <dataset-name>tuple1</dataset-name>
        <event-type-list>
            <event-type>TupleEvent1</event-type>
        </event-type-list>
    <provider-name>test-rdbms-provider</provider-name>
</playback-parameters>
```

C.60 playback-speed

Use this element to define the playback speed as a positive float. The default value is 1, which corresponds to normal speed. A value of 2 means that events will be played back 2 times faster than the original record speed. Similarly, a value of 0.5 means that events will be played back at half the speed.

C.60.1 Child Elements

The `playback-speed` component configuration element has no child elements.

C.60.2 Attributes

The `playback-speed` component has no attributes.

C.60.3 Example

The following example shows how to use the `duration` element in the component configuration file:

```
<playback-parameters>
    <dataset-name>tuple1</dataset-name>
    <event-type-list>
        <event-type>TupleEvent1</event-type>
    </event-type-list>
    <provider-name>test-rdbms-provider</provider-name>
    <store-policy-parameters>
        <parameter>
            <name>timeout</name>
            <value>300</value>
        <parameter>
    </store-policy-parameters>
    <time-range-offset>
        <start>04-07-2009:06:00:00</start>
        <duration>03:00:00</duration>
    </time-range-offset>
    <playback-speed>100</playback-speed>
</playback-parameters>
```

C.61 processor (EPL)

Use this element to define an Oracle CQL or EPL processor component.

For more information, see Chapter 11, "Configuring EPL Processors".

For information on the processor element for Oracle CQL processors, see processor (Oracle CQL).

C.61.1 Child Elements

The `processor` component configuration element supports the following child elements:

- name
- record-parameters
- playback-parameters
- rules
- database
- bindings

C.61.2 Attributes

The `processor` component has no attributes.

C.61.3 Example

The following example shows how to use the `processor` element in the component configuration file:

```
<processor>
    <name>processor1</name>
        <record-parameters>
            <dataset-name>test1data</dataset-name>
            <event-type-list>
```

```
                    <event-type>SimpleEvent</event-type>
            </event-type-list>
            <provider-name>test-rdbms-provider</provider-name>
            <batch-size>1</batch-size>
            <batch-time-out>10</batch-time-out>
        </record-parameters>
        <rules>
            <rule id="rule1"><![CDATA[
                select stockSymbol, avg(price) as percentage
                from StockTick retain 5 events
                where stockSymbol=?
                having avg(price)  > ? or avg(price)  < ?
            ]]></rule>
        </rules>
        <bindings>
            <binding id="rule1">
                <params>BEAS,10.0,-10.0</params>
                <params id="IBM">IBM,5.0,5.0</params>
            </binding>
        </bindings>
</processor>
```

In the example, the processor's unique identifier is `cqlProcessor`.

C.62 processor (Oracle CQL)

Use this element to define an Oracle CQL processor component.

For more information, see Chapter 10, "Configuring Oracle CQL Processors".

For information on the `processor` element for EPL processors, see processor (EPL).

C.62.1 Child Elements

The `processor` component configuration element supports the following child elements:

- name
- record-parameters
- playback-parameters
- rules

C.62.2 Attributes

The `processor` component has no attributes.

C.62.3 Example

The following example shows how to use the `processor` element in the component configuration file:

```
<processor>
    <name>cqlProcessor</name>
    <rules>
        <view id="lastEvents" schema="cusip bid srcId bidQty ask askQty seq"><![CDATA[
            select cusip, bid, srcId, bidQty, ask, askQty, seq
            from inputChannel[partition by srcId, cusip rows 1]
        ]]></view>
        <view id="bidask" schema="cusip bid ask"><![CDATA[
            select cusip, max(bid), min(ask)
            from lastEvents
```

```
                group by cusip
        ]]></view>
            <view ...><![CDATA[
                ...
        ]]></view>
        ...
        <view id="MAXBIDMINASK" schema="cusip bidseq bidSrcId bid askseq askSrcId ask bidQty
askQty"><![CDATA[
            select bid.cusip, bid.seq, bid.srcId as bidSrcId, bid.bid, ask.seq, ask.srcId as
askSrcId, ask.ask, bid.bidQty, ask.askQty
            from BIDMAX as bid, ASKMIN as ask
            where bid.cusip = ask.cusip
        ]]></view>
        <query id="BBAQuery"><![CDATA[
            ISTREAM(select bba.cusip, bba.bidseq, bba.bidSrcId, bba.bid, bba.askseq,
bba.askSrcId, bba.ask, bba.bidQty, bba.askQty, "BBAStrategy" as intermediateStrategy, p.seq
as correlationId, 1 as priority
            from MAXBIDMINASK as bba, inputChannel[rows 1] as p where bba.cusip = p.cusip)
        ]]></query>
    </rules>
</processor>
```

In the example, the processor's unique identifier is `cqlProcessor`.

C.63 profile

Use this element to define a diagnostic profile.

For more information, see "Monitoring the Throughput and Latency of a Stage or Path in the EPN" in the *Oracle CEP Visualizer User's Guide*.

C.63.1 Child Elements

The `profile` component configuration element supports the following child elements:

- name
- enabled
- start-stage
- max-latency
- average-latency
- throughput

C.63.2 Attributes

The `profile` component has no attributes.

C.63.3 Example

The following example shows how to use the `profile` element in the component configuration file:

```
<diagnostic-profiles>
    <name>myselfprofiles</name>
    <profile>
        <name>testProfile0</name>
        <enabled>true</enabled>
        <start-stage>MetricSubscriber</start-stage>
        <max-latency>
```

```
                <start-location>
                    <application>diagnostic</application>
                    <stage>MetricSubscriber</stage>
                    <direction>INBOUND</direction>
                </start-location>
                <end-location>
                    <application>diagnostic</application>
                    <stage>MonitorProcessor</stage>
                    <direction>OUTBOUND</direction>
                </end-location>
            </max-latency>
            <average-latency>
                <start-location>
                    <application>diagnostic</application>
                    <stage>MetricSubscriber</stage>
                    <direction>INBOUND</direction>
                </start-location>
                <end-location>
                    <application>diagnostic</application>
                    <stage>MonitorProcessor</stage>
                    <direction>OUTBOUND</direction>
                </end-location>
                <threshhold>
                    <amount>100</amount>
                    <unit>MILLISECONDS</unit>
                </threshhold>
            </average-latency>
            <throughput>
                <throughput-interval>
                    <amount>100000</amount>
                    <unit>MICROSECONDS</unit>
                </throughput-interval>
                <average-interval>
                    <amount>100000000</amount>
                    <unit>NANOSECONDS</unit>
                </average-interval>
                <location>
                    <application>diagnostic</application>
                    <stage>AlertEventStream</stage>
                    <direction>INBOUND</direction>
                </location>
            </throughput>
        </profile>
    </diagnostic-profiles>
```

C.64 provider-name

Use the `provider-name` element in the following parent elements:

- netio: Use the `provider-name` element to define which provider to use for the underlying socket implementation. Valid value is an Oracle CEP server `config.xml` file `netio` child element `provider-type`.

- record-parameters: Use the `provider-name` element to define the name of the event store provider. The value of this element corresponds to the value of the `name` child element of the `rdbms-event-store-provider` element in the `config.xml` file of the Oracle CEP server instance. When configuring the Oracle RDBMS-based provider, you are required to specify this element; see Section 13.2.1, "Configuring an Event Store for Oracle CEP Server."

C.64.1 Child Elements

The `provider-name` component configuration element has no child elements.

C.64.2 Attributes

The `provider-name` component has no attributes.

C.64.3 Example

The following example shows how to use the `provider-name` element in the component configuration file:

```
<netio>
        <provider-name>providerCache</provider-name>
        <num-threads>1000</num-threads>
</netio>
```

C.65 query

Use this element to define an Oracle CQL query for a component.

For more information, see Chapter 10, "Configuring Oracle CQL Processors".

C.65.1 Child Elements

The `query` component configuration element has no child elements.

C.65.2 Attributes

Table C–5 lists the attributes of the `query` component configuration element.

Table C–5   Attributes of the query Component Configuration Element

Attribute	Description	Data Type	Required?
`id`	Unique identifier for this query.	`String`	Yes.
`active`	Execute this query when the application is deployed and run. Valid values are `true` and `false`. Default value is `false`.	`Boolean`	No.

C.65.3 Example

The following example shows how to use the `query` element in the component configuration file:

```
<processor>
    <name>cqlProcessor</name>
    <rules>
        <view id="lastEvents" schema="cusip bid srcId bidQty ask askQty seq"><![CDATA[
            select cusip, bid, srcId, bidQty, ask, askQty, seq
            from inputChannel[partition by srcId, cusip rows 1]
        ]]></view>
        <view id="bidask" schema="cusip bid ask"><![CDATA[
            select cusip, max(bid), min(ask)
            from lastEvents
            group by cusip
        ]]></view>
            <view ...><![CDATA[
                ...
        ]]></view>
        ...
        <view id="MAXBIDMINASK" schema="cusip bidseq bidSrcId bid askseq askSrcId ask bidQty
askQty"><![CDATA[
            select bid.cusip, bid.seq, bid.srcId as bidSrcId, bid.bid, ask.seq, ask.srcId as
```

```
askSrcId, ask.ask, bid.bidQty, ask.askQty
            from BIDMAX as bid, ASKMIN as ask
            where bid.cusip = ask.cusip
        ]]></view>
        <query id="BBAQuery"><![CDATA[
            ISTREAM(select bba.cusip, bba.bidseq, bba.bidSrcId, bba.bid, bba.askseq,
bba.askSrcId, bba.ask, bba.bidQty, bba.askQty, "BBAStrategy" as intermediateStrategy, p.seq
as correlationId, 1 as priority
            from MAXBIDMINASK as bba, inputChannel[rows 1] as p where bba.cusip = p.cusip)
        ]]></query>
    </rules>
</processor>
```

C.66 record-parameters

Use this element to define event record parameters for a component.

For more information, see Chapter 13, "Configuring Event Record and Playback".

C.66.1 Child Elements

The `record-parameters` component configuration element supports the following child elements:

- dataset-name

- event-type-list

- provider-name

- store-policy-parameters

- max-size

- max-threads

- One of:

    - time-range

    - time-range-offset

- batch-size

- batch-time-out

C.66.2 Attributes

The `record-parameters` component has no attributes.

C.66.3 Example

The following example shows how to use the `record-parameters` element in the component configuration file:

```
<record-parameters>
    <dataset-name>tuple1</dataset-name>
        <event-type-list>
            <event-type>TupleEvent1</event-type>
        </event-type-list>
        <provider-name>test-rdbms-provider</provider-name>
    <batch-size>1</batch-size>
    <batch-time-out>10</batch-time-out>
</record-parameters>
```

C.67 repeat

Use this element to define whether or not to repeat playback-parameters.

Valid values are:

- true

- false

C.67.1 Child Elements

The repeat component configuration element has no child elements.

C.67.2 Attributes

The repeat component has no attributes.

C.67.3 Example

The following example shows how to use the duration element in the component configuration file:

```
<playback-parameters>
    <dataset-name>tuple1</dataset-name>
    <event-type-list>
        <event-type>TupleEvent1</event-type>
    </event-type-list>
    <provider-name>test-rdbms-provider</provider-name>
    <store-policy-parameters>
        <parameter>
            <name>timeout</name>
            <value>300</value>
        <parameter>
    </store-policy-parameters>
    <time-range-offset>
        <start>04-07-2009:06:00:00</start>
        <duration>03:00:00</duration>
    </time-range-offset>
    <repeat>true</repeat>
</playback-parameters>
```

C.68 rule

Use this element to define an EPL rule for a component.

This element is applicable only in a rules element.

C.68.1 Child Elements

The rule component configuration element has no child elements.

C.68.2 Attributes

Table C–6 lists the attributes of the rule component configuration element.

Table C–6   Attributes of the rule Component Configuration Element

Attribute	Description	Data Type	Required?
id	Unique identifier for this rule.	String	Yes.

Table C–6   (Cont.) Attributes of the rule Component Configuration Element

Attribute	Description	Data Type	Required?
`active`	Execute this rule when the application is deployed and run. Valid values are `true` and `false`. Default value is `false`.	`Boolean`	No.

C.68.3 Example

The following example shows how to use the `rule` element in the component configuration file:

```
<processor>
    <name>rvSampleProcessor</name>
    <rules>
        <rule id="rvSampleRule1"><![CDATA[
            select * from RVSampleEvent retain 1 event
        ]]></rule>
    </rules>
</processor>
```

C.69 rules

Use this element to define one or more Oracle CQL queries or views for a processor (Oracle CQL) or EPL rules for a processor (EPL).

C.69.1 Child Elements

The `rules` component configuration element supports the following child elements:

- rule

- query

- view

C.69.2 Attributes

The `rules` component has no attributes.

C.69.3 Example

The following example shows how to use the `rules` element in the component configuration file:

```
<processor>
    <name>cqlProcessor</name>
    <rules>
        <view id="lastEvents" schema="cusip bid srcId bidQty ask askQty seq"><![CDATA[
            select cusip, bid, srcId, bidQty, ask, askQty, seq
            from inputChannel[partition by srcId, cusip rows 1]
        ]]></view>
        <view id="bidask" schema="cusip bid ask"><![CDATA[
            select cusip, max(bid), min(ask)
            from lastEvents
            group by cusip
        ]]></view>
            <view ...><![CDATA[
                ...
        ]]></view>
        ...
        <view id="MAXBIDMINASK" schema="cusip bidseq bidSrcId bid askseq askSrcId ask bidQty
```

```
askQty"><![CDATA[
            select bid.cusip, bid.seq, bid.srcId as bidSrcId, bid.bid, ask.seq, ask.srcId as
askSrcId, ask.ask, bid.bidQty, ask.askQty
            from BIDMAX as bid, ASKMIN as ask
            where bid.cusip = ask.cusip
        ]]></view>
        <query id="BBAQuery"><![CDATA[
            ISTREAM(select bba.cusip, bba.bidseq, bba.bidSrcId, bba.bid, bba.askseq,
bba.askSrcId, bba.ask, bba.bidQty, bba.askQty, "BBAStrategy" as intermediateStrategy, p.seq
as correlationId, 1 as priority
            from MAXBIDMINASK as bba, inputChannel[rows 1] as p where bba.cusip = p.cusip)
        ]]></query>
    </rules>
</processor>
```

C.70 schedule-time-range

Use this element to define the time during which events will be played back to the stage. Playing back will start at the specified start time and will continue until all the events are played back or specified end time. If repeat is set to true, playback will continue until the specified end time or until playback is explicitly stopped by the user.

This element is applicable only to the playback-parameters element.

C.70.1 Child Elements

The `schedule-time-range` component configuration element supports the following child elements:

- start

- end

C.70.2 Attributes

The `schedule-time-range` component has no attributes.

C.70.3 Example

The following example shows how to use the `schedule-time-range` element in the component configuration file:

```
<record-parameters>
    <dataset-name>tuple1</dataset-name>
    <event-type-list>
        <event-type>TupleEvent1</event-type>
    </event-type-list>
    <provider-name>test-rdbms-provider</provider-name>
    <store-policy-parameters>
        <parameter>
            <name>timeout</name>
            <value>300</value>
        </parameter>
    </store-policy-parameters>
    <schedule-time-range>
        <start>04-07-2009:06:00:00</start>
        <end>04-10-2009:22:00:00</end>
    </schedule-time-range>
    <batch-size>1</batch-size>
    <batch-time-out>10</batch-time-out>
</record-parameters>
```

C.71 schedule-time-range-offset

Use this element to define the time during which events will be played back to the stage. Playing back will start at the specified start time and will continue until all the events are played back or specified end time. If repeat is set to true, playback will continue until the specified end time or until playback is explicitly stopped by the user.

This element is applicable only to the playback-parameters element.

C.71.1 Child Elements

The schedule-time-range-offset component configuration element supports the following child elements:

- start

- duration

C.71.2 Attributes

The schedule-time-range-offset component has no attributes.

C.71.3 Example

The following example shows how to use the schedule-time-range-offset element in the component configuration file:

```
<record-parameters>
    <dataset-name>tuple1</dataset-name>
    <event-type-list>
        <event-type>TupleEvent1</event-type>
    </event-type-list>
    <provider-name>test-rdbms-provider</provider-name>
    <store-policy-parameters>
        <parameter>
            <name>timeout</name>
            <value>300</value>
        <parameter>
    </store-policy-parameters>
    <schedule-time-range-offset>
        <start>04-07-2009:06:00:00</start>
        <duration>03:00:00</duration>
    </schedule-time-range-offset>
    <batch-size>1</batch-size>
    <batch-time-out>10</batch-time-out>
</record-parameters>
```

C.72 selector

Use this element to specify which up-stream Oracle CQL processor queries are permitted to output their results to a downstream channel.

Figure C–1 shows an EPN with channel filteredStream connected to up-stream Oracle CQL processor filteredFanoutProcessor.

Figure C–1   EPN With Oracle CQL Processor and Down-Stream Channel

PriceAdapter   priceStream   filterFanoutProcessor   filteredStream
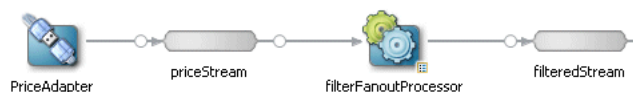
Example C–12 shows the queries configured for the Oracle CQL processor.

Example C–12   filterFanoutProcessor Oracle CQL Queries

```
<processor>
    <name>filterFanoutProcessor</name>
    <rules>
        <query id="Yr3Sector"><![CDATA[
            select cusip, bid, srcId, bidQty, ask, askQty, seq
            from priceStream where sector="3_YEAR"
        ]]></query>
        <query id="Yr2Sector"><![CDATA[
            select cusip, bid, srcId, bidQty, ask, askQty, seq
            from priceStream where sector="2_YEAR"
        ]]></query>
        <query id="Yr1Sector"><![CDATA[
            select cusip, bid, srcId, bidQty, ask, askQty, seq
            from priceStream where sector="1_YEAR"
        ]]></query>
    </rules>
</processor>
```

If you specify more than one query for an Oracle CQL processor as Example C–12 shows, then, by default, all query results are output to the processor's out-bound channel (filteredStream in Figure C–1). Optionally, in the component configuration source, you can use the channel element selector attribute to specify a space-delimited list of one or more Oracle CQL query names that may output their results to the channel as Example C–13 shows. In this example, query results for query Yr3Sector and Yr2Sector are output to filteredStream but not query results for query Yr1Sector.

Example C–13   Using selector to Control Which Query Results are Output

```
<channel>
    <name>filteredStream</name>
    <selector>Yr3Sector Yr2Sector</selector>
</channel>
```

> **Note:** The selector attribute is only applicable if the up-stream node is an Oracle CQL processor. For more information, see Chapter 10, "Configuring Oracle CQL Processors".

C.72.1 Child Elements

The selector component configuration element has no child elements.

C.72.2 Attributes

The selector component has no attributes.

C.72.3 Example

The following example shows how to use the `selector` element in the component configuration file:

```
<processor>
    <name>PatternProcessor</name>
    <rules>
        <query id="match"><![CDATA[
            select T.firstW as firstw, T.lastZ as lastz, T.price as price
            from StockInputsStream
            MATCH_RECOGNIZE (
                MEASURES A.c1 as firstW, last(Z.c1) as lastZ, A.c2 as price
                PATTERN(A W+ X+ Y+ Z+)
                DEFINE  A as A.c1 = A.c1,
                        W as W.c2 < prev(W.c2),
                        X as X.c2 > prev(X.c2),
                        Y as Y.c2 < prev(Y.c2),
                        Z as Z.c2 > prev(Z.c2))
            as T
        ]]></query>
         <query id="stock"><![CDATA[
            select c1 as ts, c2 as price from StockInputsStream
        ]]></query>
    </rules>
</processor>
<channel>
    <name>StockOutputChannel</name>
    <selector>stock</selector>
</channel>
<channel>
    <name>MatchOutputChannel</name>
    <selector>match</selector>
</channel>
```

C.73 server-context-path

Use this element to define the path of the local HTTP pub-sub server associated with the Oracle CEP instance hosting the current Oracle CEP application, for each *local* `http-pub-sub-adapter` configured for publishing.

By default, each Oracle CEP server is configured with an HTTP pub-sub server with path `/pubsub`; if, however, you have created a new local HTTP pub-sub server, or changed the default configuration, then specify the value of the `path` child element of the `http-pubsub` element in the Oracle CEP server `config.xml` file.

C.73.1 Child Elements

The `server-context-path` component configuration element has no child elements.

C.73.2 Attributes

The `server-context-path` component has no attributes.

C.73.3 Example

The following example shows how to use the `server-context-path` element in the component configuration file:

```
<http-pub-sub-adapter>
    <name>localPub</name>
```

```
        <server-context-path>/pubsub</server-context-path>
        <channel>/test1</channel>
</http-pub-sub-adapter>
```

C.74 server-url

Use this element to define the URL of the *remote* HTTP pub-sub server to which the
Oracle CEP application will publish or subscribe, respectively. The remote pub-sub
server could be another instance of Oracle CEP, or a WebLogic Server instance, or it
could be any third-party HTTP pub-sub server (for both publishing and subscribing).

C.74.1 Child Elements

The `server-url` component configuration element has no child elements.

C.74.2 Attributes

The `server-url` component has no attributes.

C.74.3 Example

The following example shows how to use the `server-url` element in the component
configuration file:

```
<http-pub-sub-adapter>
    <name>remotePub</name>
    <server-url>http://myhost.com:9102/pubsub</server-url>
    <channel>/test1</channel>
    <event-type>com.bea.wlevs.tests.httppubsub.PubsubTestEvent</event-type>
    <user>wlevs</user>
    <password>wlevs</password>
</http-pub-sub-adapter>
```

In the example, the URL of the remote HTTP pub-sub server to which the
`remotePublisher` adapter will publish events is
`http://myhost.com:9102/pubsub`.

C.75 session-ack-mode-name

Use this element to define the session acknowledge mode name for a jms-adapter.

Valid values from javax.jms.Session are:

- `AUTO_ACKNOWLEDGE`: With this acknowledgment mode, the session automatically
  acknowledges a client's receipt of a message either when the session has
  successfully returned from a call to receive or when the message listener the
  session has called to process the message successfully returns.

- `CLIENT_ACKNOWLEDG`: With this acknowledgment mode, the client acknowledges
  a consumed message by calling the message's acknowledge method.

- `DUPS_OK_ACKNOWLEDGE`: This acknowledgment mode instructs the session to
  lazily acknowledge the delivery of messages.

C.75.1 Child Elements

The `session-ack-mode-name` component configuration element has no child
elements.

C.75.2 Attributes

The `session-ack-mode-name` component has no attributes.

C.75.3 Example

The following example shows how to use the `session-ack-mode-name` element in the component configuration file:

```
<jms-adapter>
    <name>jmsInbound-text</name>
    <connection-jndi-name>weblogic.jms.ConnectionFactory</connection-jndi-name>
    <destination-name>JMSServer-0/Module1!Queue1</destination-name>
    <user>weblogic</user>
    <password>weblogic</password>
    <work-manager>JettyWorkManager</work-manager>
    <session-ack-mode-name>AUTO_ACKNOWLEDGE</session-ack-mode-name>
    <session-transacted>false</session-transacted>
</jms-adapter>
```

C.76 session-transacted

Use this element to define whether or not a session is transacted for a jms-adapter.

Valid values are:

- `true`
- `false`

C.76.1 Child Elements

The `session-transacted` component configuration element has no child elements.

C.76.2 Attributes

The `session-transacted` component has no attributes.

C.76.3 Example

The following example shows how to use the `session-transacted` element in the component configuration file:

```
<jms-adapter>
    <name>jmsInbound-text</name>
    <connection-jndi-name>weblogic.jms.ConnectionFactory</connection-jndi-name>
    <destination-name>JMSServer-0/Module1!Queue1</destination-name>
    <user>weblogic</user>
    <password>weblogic</password>
    <work-manager>JettyWorkManager</work-manager>
    <session-ack-mode-name>AUTO_ACKNOWLEDGE</session-ack-mode-name>
    <session-transacted>false</session-transacted>
</jms-adapter>
```

C.77 stage

Use this element to define the stage for a start-location or end-location element of a diagnostic profile.

Valid values are the name of an existing stage in your Event Processing Network (EPN). For more information, see Section 1.1.1, "Components of the Oracle CEP Event Processing Network".

C.77.1 Child Elements

The stage component configuration has no child elements:

C.77.2 Attributes

The stage component has no attributes.

C.77.3 Example

The following example shows how to use the stage element in the component configuration file:

```
<diagnostic-profiles>
    <name>myselfprofiles</name>
    <profile>
        <name>testProfile0</name>
        <enabled>true</enabled>
        <start-stage>MetricSubscriber</start-stage>
        <max-latency>
            <collect-interval>
                <amount>1000</amount>
                <unit>s</unit>
            </collect-interval>
            <name>testProfile0MaxLat</name>
            <start-location>
                <application>diagnostic</application>
                <stage>MetricSubscriber</stage>
                <direction>INBOUND</direction>
            </start-location>
            <end-location>
                <application>diagnostic</application>
                <stage>MonitorProcessor</stage>
                <direction>OUTBOUND</direction>
            </end-location>
        </max-latency>
    </profile>
</diagnostic-profiles>
```

C.78 start

Use this element to define a start time for a time-range element using the following format.

```
MM-DD-YYYY:HH:MM:SS
```

Where MM is the two-digit month, DD is the two-digit day of the month, YYYY is the four-digit year, HH is the hour (in 24-hour format), MM is the two-digit minutes, and SS is the two-digit seconds.

C.78.1 Child Elements

The start component configuration element has no child elements.

C.78.2 Attributes

The start component has no attributes.

C.78.3 Example

The following example shows how to use the start element in the component configuration file:

```
<record-parameters>
    <dataset-name>tuple1</dataset-name>
    <event-type-list>
        <event-type>TupleEvent1</event-type>
    </event-type-list>
    <provider-name>test-rdbms-provider</provider-name>
    <store-policy-parameters>
        <parameter>
            <name>timeout</name>
            <value>300</value>
        <parameter>
    </store-policy-parameters>
    <time-range>
        <start>04-07-2009:06:00:00</start>
        <end>04-10-2009:22:00:00</end>
    </time-range>
    <batch-size>1</batch-size>
    <batch-time-out>10</batch-time-out>
</record-parameters>
```

C.79 start-location

Use this element to define the start location of a diagnostic profile.

For more information, see "Monitoring the Throughput and Latency of a Stage or Path in the EPN" in the *Oracle CEP Visualizer User's Guide*.

C.79.1 Child Elements

The start-location component configuration element supports the following child elements:

- application

- stage

- direction

C.79.2 Attributes

The start-location component has no attributes.

C.79.3 Example

The following example shows how to use the start-location element in the component configuration file:

```
<diagnostic-profiles>
    <name>myselfprofiles</name>
    <profile>
        <name>testProfile0</name>
        <enabled>true</enabled>
        <start-stage>MetricSubscriber</start-stage>
        <max-latency>
            <collect-interval>
                <amount>1000</amount>
                <unit>s</unit>
            </collect-interval>
```

```
                    <name>testProfile0MaxLat</name>
                    <start-location>
                        <application>diagnostic</application>
                        <stage>MetricSubscriber</stage>
                        <direction>INBOUND</direction>
                    </start-location>
                    <end-location>
                        <application>diagnostic</application>
                        <stage>MonitorProcessor</stage>
                        <direction>OUTBOUND</direction>
                    </end-location>
                </max-latency>
            </profile>
        </diagnostic-profiles>
```

C.80 start-stage

Use this element to define the starting stage of a diagnostic profile.

Valid values are the name of an existing stage in your Event Processing Network (EPN). For more information, see Section 1.1.1, "Components of the Oracle CEP Event Processing Network".

For more information, see "Monitoring the Throughput and Latency of a Stage or Path in the EPN" in the *Oracle CEP Visualizer User's Guide*.

C.80.1 Child Elements

The start-stage component configuration element has no child elements.

C.80.2 Attributes

The start-stage component has no attributes.

C.80.3 Example

The following example shows how to use the start-stage element in the component configuration file:

```
<diagnostic-profiles>
    <name>myselfprofiles</name>
    <profile>
        <name>testProfile0</name>
        <enabled>true</enabled>
        <start-stage>MetricSubscriber</start-stage>
        <max-latency>
            <start-location>
                <application>diagnostic</application>
                <stage>MetricSubscriber</stage>
                <direction>INBOUND</direction>
            </start-location>
            <end-location>
                <application>diagnostic</application>
                <stage>MonitorProcessor</stage>
                <direction>OUTBOUND</direction>
            </end-location>
        </max-latency>
    </profile>
</diagnostic-profiles>
```

C.81 store-policy-parameters

Use this element to define one or more store policy parameter, specific to the event store provider.

C.81.1 Child Elements

The `store-policy-parameter` component configuration element supports the following child elements:

- parameter

C.81.2 Attributes

The `store-policy-parameter` component has no attributes.

C.81.3 Example

The following example shows how to use the `store-policy-parameter` element in the component configuration file:

```
<record-parameters>
    <dataset-name>tuple1</dataset-name>
        <event-type-list>
            <event-type>TupleEvent1</event-type>
        </event-type-list>
        <provider-name>test-rdbms-provider</provider-name>
        <store-policy-parameters>
            <parameter>
                <name>timeout</name>
                <value>300</value>
            <parameter>
        </store-policy-parameters>
    <batch-size>1</batch-size>
    <batch-time-out>10</batch-time-out>
</record-parameters>
```

C.82 stream

Use this element to define a stream component.

> **Note:** The `stream` component is deprecated in Release 11*g*R1 (11.1.1). Use the channel element instead.

C.82.1 Child Elements

The `stream` component configuration element supports the following child elements:

- name

- record-parameters

- playback-parameters

- max-size

- max-threads

C.82.2 Attributes

The `stream` component has no attributes.

C.82.3 Example

The following example shows how to use the `stream` element in the component configuration file:

```
<stream>
    <name>fxMarketEuroOut</name>
    <max-size>0</max-size>
    <max-threads>0</max-threads>
</stream>
```

In the example, the stream's unique identifier is `fxMarketEuroOut`.

C.83 symbol

Use this element to define a symbol for an adapter, http-pub-sub-adapter, or jms-adapter element.

> **Note:** The `symbol` component is deprecated in Release 11*g*R1 (11.1.1).

C.83.1 Child Elements

The `symbol` component configuration has no child elements:

C.83.2 Attributes

The `symbol` component has no attributes.

C.83.3 Example

The following example shows how to use the `symbol` element in the component configuration file:

```
<adapter>
    <name>trackdata</name>
    <symbols>
        <symbol>BEAS</symbol>
        <symbol>IBM</symbol>
    </symbols>
</adapter>
```

C.84 symbols

Use this element to define one or more symbol elements for a component.

> **Note:** The `symbol` component is deprecated in Release 11*g*R1 (11.1.1).

C.84.1 Child Elements

The `symbols` component configuration element supports the following child elements:

- symbol

C.84.2 Attributes

The `symbols` component has no attributes.

C.84.3 Example

The following example shows how to use the `symbols` element in the component configuration file:

```
<adapter>
    <name>trackdata</name>
    <symbols>
        <symbol>BEAS</symbol>
        <symbol>IBM</symbol>
    </symbols>
</adapter>
```

C.85 threshhold

Use this element to define the threshold above which Oracle CEP server logs a monitoring event.

This element is applicable only in an average-latency element in a diagnostic profile.

For more information, see "Monitoring the Throughput and Latency of a Stage or Path in the EPN" in the *Oracle CEP Visualizer User's Guide*.

C.85.1 Child Elements

The `threshhold` component configuration element supports the following child elements:

- amount
- unit

C.85.2 Attributes

The `threshhold` component has no attributes.

C.85.3 Example

The following example shows how to use the `threshhold` element in the component configuration file:

```
<diagnostic-profiles>
    <name>myselfprofiles</name>
    <profile>
        <name>testProfile0</name>
        <enabled>true</enabled>
        <start-stage>MetricSubscriber</start-stage>
        <average-latency>
            <start-location>
                <application>diagnostic</application>
```

```
                        <stage>MetricSubscriber</stage>
                        <direction>INBOUND</direction>
                    </start-location>
                    <end-location>
                        <application>diagnostic</application>
                        <stage>MonitorProcessor</stage>
                        <direction>OUTBOUND</direction>
                    </end-location>
                    <threshhold>
                        <amount>100</amount>
                        <unit>MILLISECONDS</unit>
                    </threshhold>
                </average-latency>
            </profile>
        </diagnostic-profiles>
```

C.86 throughput

Use this element to define a throughput diagnostic profile.

For more information, see "Monitoring the Throughput and Latency of a Stage or Path in the EPN" in the *Oracle CEP Visualizer User's Guide*.

C.86.1 Child Elements

The throughput component configuration element supports the following child elements:

- name

- throughput-interval

- average-interval

- location

C.86.2 Attributes

The throughput component has no attributes.

C.86.3 Example

The following example shows how to use the throughput element in the component configuration file:

```
<diagnostic-profiles>
    <name>myselfprofiles</name>
    <profile>
        <name>testProfile0</name>
        <enabled>true</enabled>
        <start-stage>MetricSubscriber</start-stage>
        <throughput>
            <throughput-interval>
                <amount>100000</amount>
                <unit>MICROSECONDS</unit>
            </throughput-interval>
            <average-interval>
                <amount>100000000</amount>
                <unit>NANOSECONDS</unit>
            </average-interval>
            <location>
                <application>diagnostic</application>
                <stage>AlertEventStream</stage>
                <direction>INBOUND</direction>
```

```
            </location>
        </throughput>
    </profile>
</diagnostic-profiles>
```

C.87 throughput-interval

Use this element to define the throughput interval of a diagnostic profile.

For more information, see "Monitoring the Throughput and Latency of a Stage or Path in the EPN" in the *Oracle CEP Visualizer User's Guide*.

C.87.1 Child Elements

The `throughput-interval` component configuration element supports the following child elements:

- amount

- unit

C.87.2 Attributes

The `throughput-interval` component has no attributes.

C.87.3 Example

The following example shows how to use the `throughput-interval` element in the component configuration file:

```
<diagnostic-profiles>
    <name>myselfprofiles</name>
    <profile>
        <name>testProfile0</name>
        <enabled>true</enabled>
        <start-stage>MetricSubscriber</start-stage>
        <throughput>
            <throughput-interval>
                <amount>100000</amount>
                <unit>MICROSECONDS</unit>
            </throughput-interval>
            <average-interval>
                <amount>100000000</amount>
                <unit>NANOSECONDS</unit>
            </average-interval>
            <location>
                <application>diagnostic</application>
                <stage>AlertEventStream</stage>
                <direction>INBOUND</direction>
            </location>
        </throughput>
    </profile>
</diagnostic-profiles>
```

C.88 time-range

Use this element to define a filter that Oracle CEP server applies to the events in the event store. Only events with a record-time in this time range will be played back to the stage.

Use either time-range-offset or `time-range` but not both.

For more information, see Chapter 13, "Configuring Event Record and Playback".

C.88.1 Child Elements

The `time-range` component configuration element supports the following child elements:

- start
- end

C.88.2 Attributes

The `time-range` component has no attributes.

C.88.3 Example

The following example shows how to use the `time-range` element in the component configuration file:

```
<record-parameters>
    <dataset-name>tuple1</dataset-name>
    <event-type-list>
        <event-type>TupleEvent1</event-type>
    </event-type-list>
    <provider-name>test-rdbms-provider</provider-name>
    <store-policy-parameters>
        <parameter>
            <name>timeout</name>
            <value>300</value>
        <parameter>
    </store-policy-parameters>
    <time-range>
        <start>04-07-2009:06:00:00</start>
        <end>04-10-2009:22:00:00</end>
    </time-range>
    <batch-size>1</batch-size>
    <batch-time-out>10</batch-time-out>
</record-parameters>
```

C.89 time-range-offset

Use this element to define a filter that Oracle CEP server applies to the events in the event store. Only events with a record-time in this time range will be played back to the stage.

Use either time-range or `time-range-offset` but not both.

For more information, see Chapter 13, "Configuring Event Record and Playback".

C.89.1 Child Elements

The `time-range-offset` component configuration element supports the following child elements:

- start
- duration

C.89.2 Attributes

The `time-range-offset` component has no attributes.

C.89.3 Example

The following example shows how to use the `time-range-offset` element in the component configuration file:

```
<record-parameters>
    <dataset-name>tuple1</dataset-name>
    <event-type-list>
        <event-type>TupleEvent1</event-type>
    </event-type-list>
    <provider-name>test-rdbms-provider</provider-name>
    <store-policy-parameters>
        <parameter>
            <name>timeout</name>
            <value>300</value>
        <parameter>
    </store-policy-parameters>
    <time-range-offset>
        <start>04-07-2009:06:00:00</start>
        <duration>03:00:00</duration>
    </time-range-offset>
    <batch-size>1</batch-size>
    <batch-time-out>10</batch-time-out>
</record-parameters>
```

C.90 time-to-live

Use this element to define the maximum amount of time, in milliseconds, that an entry is cached. Default value is infinite.

For more information, see Section 12.2, "Configuring an Oracle CEP Local Caching System and Cache".

C.90.1 Child Elements

The `time-to-live` component configuration element has no child elements.

C.90.2 Attributes

The `time-to-live` component has no attributes.

C.90.3 Example

The following example shows how to use the `time-to-live` element in the component configuration file:

```
<caching-system>
    <name>providerCachingSystem</name>
    <cache>
        <name>providerCache</name>
        <max-size>1000</max-size>
        <eviction-policy>FIFO</eviction-policy>
        <time-to-live>60000</time-to-live>
        <idle-time>120000</idle-time>
        <write-none/>
        <work-manager-name>JettyWorkManager</work-manager-name>
        <listeners asynchronous="false">
```

```
            <work-manager-name>JettyWorkManager</work-manager-name>
        </listeners>
    </cache>
</caching-system>
```

C.91 unit

Use this element to define the duration units of amount element.

Valid values are:

- NANOSECONDS

- MICROSECONDS

- MILLISECONDS

- SECONDS

- MINUTES

- HOURS

- DAYS

C.91.1 Child Elements

The unit component configuration has no child elements:

C.91.2 Attributes

The unit component has no attributes.

C.91.3 Example

The following example shows how to use the unit element in the component configuration file:

```
<diagnostic-profiles>
    <name>myselfprofiles</name>
    <profile>
        <name>testProfile0</name>
        <enabled>true</enabled>
        <start-stage>MetricSubscriber</start-stage>
        <max-latency>
            <collect-interval>
                <amount>1000</amount>
                <unit>s</unit>
            </collect-interval>
            <name>testProfile0MaxLat</name>
            <start-location>
                <application>diagnostic</application>
                <stage>MetricSubscriber</stage>
                <direction>INBOUND</direction>
            </start-location>
            <end-location>
                <application>diagnostic</application>
                <stage>MonitorProcessor</stage>
                <direction>OUTBOUND</direction>
            </end-location>
        </max-latency>
    </profile>
</diagnostic-profiles>
```

C.92 user

Use the `user` element in the following parent elements:

- http-pub-sub-adapter: Use the `user` element to define the user name if the HTTP pub-sub server to which the Oracle CEP application is publishing requires user authentication.

- jms-adapter: Use the `user` element to define the user name if the JMS service provider requires user authentication.

C.92.1 Child Elements

The `user` component configuration element has no child elements.

C.92.2 Attributes

The `user` component has no attributes.

C.92.3 Example

The following example shows how to use the `user` element in the component configuration file:

```
<http-pub-sub-adapter>
    <name>remotePub</name>
    <server-url>http://localhost:9002/pubsub</server-url>
    <channel>/test1</channel>
    <event-type>com.bea.wlevs.tests.httppubsub.PubsubTestEvent</event-type>
    <user>wlevs</user>
    <password>wlevs</password>
</http-pub-sub-adapter>
```

C.93 value

Use this element to define the value of a parameter element.

C.93.1 Child Elements

The `value` component configuration element has no child elements.

C.93.2 Attributes

The `value` component has no attributes.

C.93.3 Example

The following example shows how to use the `value` element in the component configuration file:

```
<record-parameters>
    <dataset-name>tuple1</dataset-name>
        <event-type-list>
            <event-type>TupleEvent1</event-type>
        </event-type-list>
        <provider-name>test-rdbms-provider</provider-name>
        <store-policy-parameters>
            <parameter>
                <name>timeout</name>
                <value>300</value>
            <parameter>
```

```
        </store-policy-parameters>
    <batch-size>1</batch-size>
    <batch-time-out>10</batch-time-out>
</record-parameters>
```

C.94 view

Use this element to define an Oracle CQL view for a component.

For more information, see Chapter 10, "Configuring Oracle CQL Processors".

C.94.1 Child Elements

The `view` component configuration element has no child elements.

C.94.2 Attributes

Table C–7 lists the attributes of the `view` component configuration element.

Table C–7   Attributes of the view Component Configuration Element

Attribute	Description	Data Type	Required?
id	Unique identifier for this query.	String	Yes.
active	Execute this query when the application is deployed and run. Valid values are `true` and `false`. Default value is `false`.	Boolean	No.
schema	Space delimited list of stream elements used in the view.	String of space delimited tokens.	No.

C.94.3 Example

The following example shows how to use the `view` element in the component configuration file:

```
<processor>
    <name>cqlProcessor</name>
    <rules>
        <view id="lastEvents" schema="cusip bid srcId bidQty ask askQty seq"><![CDATA[
            select cusip, bid, srcId, bidQty, ask, askQty, seq
            from inputChannel[partition by srcId, cusip rows 1]
        ]]></view>
        <view id="bidask" schema="cusip bid ask"><![CDATA[
            select cusip, max(bid), min(ask)
            from lastEvents
            group by cusip
        ]]></view>
            <view ...><![CDATA[
                ...
        ]]></view>
        ...
        <view id="MAXBIDMINASK" schema="cusip bidseq bidSrcId bid askseq askSrcId ask bidQty
askQty"><![CDATA[
            select bid.cusip, bid.seq, bid.srcId as bidSrcId, bid.bid, ask.seq, ask.srcId as
askSrcId, ask.ask, bid.bidQty, ask.askQty
            from BIDMAX as bid, ASKMIN as ask
            where bid.cusip = ask.cusip
        ]]></view>
        <query id="BBAQuery"><![CDATA[
```

```
                ISTREAM(select bba.cusip, bba.bidseq, bba.bidSrcId, bba.bid, bba.askseq,
bba.askSrcId, bba.ask, bba.bidQty, bba.askQty, "BBAStrategy" as intermediateStrategy, p.seq
as correlationId, 1 as priority
                from MAXBIDMINASK as bba, inputChannel[rows 1] as p where bba.cusip = p.cusip)
        ]]></query>
    </rules>
</processor>
```

C.95 work-manager

Use this element to define the name of a work manager for a jms-adapter.

Valid value is the name specified in the Oracle CEP server `config.xml` file `work-manager` element `name` child element.

The default value is the work manager configured for the application itself.

C.95.1 Child Elements

The `work-manager` component configuration element has no child elements:

C.95.2 Attributes

The `work-manager` component has no attributes.

C.95.3 Example

The following example shows how to use the `work-manager` element in the component configuration file:

```
<jms-adapter>
    <name>jmsInbound-text</name>
    <jndi-provider-url>t3://localhost:7001</jndi-provider-url>
    <destination-name>JMSServer-0/Module1!Queue1</destination-name>
    <user>weblogic</user>
    <password>weblogic</password>
    <work-manager>JettyWorkManager</work-manager>
    <concurrent-consumers>1</concurrent-consumers>
    <session-transacted>false</session-transacted>
</jms-adapter>
```

C.96 work-manager-name

Use this element to define a work manager for a cache.

The listeners element has a single child element, work-manager-name, that specifies the work manager to be used for asynchronously invoking listeners. This value is ignored if synchronous invocations are enabled. If a work manager is specified for the cache itself, this value overrides it for invoking listeners only.

Valid value is the name specified in the Oracle CEP server `config.xml` file `work-manager` element `name` child element.

The default value is the work manager configured for the application itself.

C.96.1 Child Elements

The `work-manager-name` component configuration element has no child elements:

C.96.2 Attributes

The `work-manager-name` component has no attributes.

C.96.3 Example

The following example shows how to use the `work-manager-name` element in the component configuration file:

```
<cache>
        <name>providerCache</name>
        <max-size>1000</max-size>
        <eviction-policy>FIFO</eviction-policy>
        <time-to-live>60000</time-to-live>
        <idle-time>120000</idle-time>
        <write-none/>
        <work-manager-name>JettyWorkManager</work-manager-name>
        <listeners asynchronous="false">
            <work-manager-name>JettyWorkManager</work-manager-name>
        </listeners>
</cache>
```

C.97 write-behind

Use this element to specify asynchronous writes to the cache store. The cache store is invoked from a separate thread after a create or update of a cache entry. This element may be changed dynamically.

C.97.1 Child Elements

The `write-behind` component configuration element supports the following child elements:

- work-manager-name

- batch-size

- buffer-size

- buffer-write-attempts

- buffer-write-timeout

C.97.2 Attributes

The `write-behind` component has no attributes.

C.97.3 Example

The following example shows how to use the `write-behind` element in the component configuration file:

```
<caching-system>
    <name>providerCachingSystem</name>
    <cache>
        <name>providerCache</name>
        <max-size>1000</max-size>
        <eviction-policy>FIFO</eviction-policy>
        <time-to-live>60000</time-to-live>
        <idle-time>120000</idle-time>
        <write-behind>
            <work-manager-name>JettyWorkManager</work-manager-name>
```

```
            <batch-size>100</batch-size>
            <buffer-size>100</buffer-size>
            <buffer-write-attempts>100</buffer-write-attempts>
            <buffer-write-timeout>100</buffer-write-timeout>
        </write-behind>
        <work-manager-name>JettyWorkManager</work-manager-name>
        <listeners asynchronous="false">
            <work-manager-name>JettyWorkManager</work-manager-name>
        </listeners>
    </cache>
</caching-system>
```

C.98 write-none

Use this element to specify no writes to a cache store. This is the default write policy. This element may be changed dynamically.

C.98.1 Child Elements

The `write-none` component configuration element has no child elements.

C.98.2 Attributes

The `write-none` component has no attributes.

C.98.3 Example

The following example shows how to use the `write-none` element in the component configuration file:

```
<caching-system>
    <name>providerCachingSystem</name>
    <cache>
        <name>providerCache</name>
        <max-size>1000</max-size>
        <eviction-policy>FIFO</eviction-policy>
        <time-to-live>60000</time-to-live>
        <idle-time>120000</idle-time>
        <write-none/>
        <work-manager-name>JettyWorkManager</work-manager-name>
        <listeners asynchronous="false">
            <work-manager-name>JettyWorkManager</work-manager-name>
        </listeners>
    </cache>
</caching-system>
```

C.99 write-through

Use this element to specify synchronous writes to the cache store. As soon as an entry is created or updated the write occurs. This element may be changed dynamically.

C.99.1 Child Elements

The `write-through` component configuration element has no child elements.

C.99.2 Attributes

The `write-through` component has no attributes.

C.99.3 Example

The following example shows how to use the `write-through` element in the component configuration file:

```
<caching-system>
    <name>providerCachingSystem</name>
    <cache>
        <name>providerCache</name>
        <max-size>1000</max-size>
        <eviction-policy>FIFO</eviction-policy>
        <time-to-live>60000</time-to-live>
        <idle-time>120000</idle-time>
        <write-through/>
        <work-manager-name>JettyWorkManager</work-manager-name>
        <listeners asynchronous="false">
            <work-manager-name>JettyWorkManager</work-manager-name>
        </listeners>
    </cache>
</caching-system>
```

D

Schema Reference: EPN Assembly spring-wlevs-v11_0_0_0.xsd

This appendix describes the elements of the `spring-wlevs-v11_0_0_0.xsd` schema.

For more information, see:

- Section D.1, "Overview of the Oracle CEP Application Assembly Elements"
- Section B.2, "EPN Assembly Schema spring-wlevs-v11_0_0_0.xsd"

D.1 Overview of the Oracle CEP Application Assembly Elements

Oracle CEP provides a number of application assembly elements 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.

D.1.1 Element Hierarchy

The Oracle CEP application assembly elements are organized into the following hierarchy:

```
beans
    Standard Spring and OSGi elements such as bean, osgi-service, and so on.
    wlevs:event-type-repository
        wlevs:event-type
            wlevs:metadata
            wlevs:property
    wlevs:adapter
        wlevs:listener
        wlevs:instance-property
        wlevs:property
    wlevs:processor
        wlevs:listener
        wlevs:source
        wlevs:function
        wlevs:instance-property
        wlevs:property
        wlevs:cache-source
        wlevs:table-source
    wlevs:channel
        wlevs:listener
        wlevs:source
        wlevs:instance-property
```

```
                        wlevs:property
                        wlevs:application-timestamped
                            wlevs:expression
                    wlevs:event-bean
                        wlevs:listener
                        wlevs:instance-property
                        wlevs:property
                    wlevs:factory
                    wlevs:cache
                        wlevs:caching-system
                        wlevs:cache-loader
                        wlevs:cache-store
                        wlevs:cache-listener
                    wlevs:caching-system
                        wlevs:instance-property
                        wlevs:property
                    wlevs:table
```

D.1.2 Example of an EPN Assembly File That Uses Oracle CEP Elements

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

```
<?xwml 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-v11_0_0_0.xsd">
    <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>
    <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:channel id="helloworldInstream" >
        <wlevs:listener ref="helloworldProcessor"/>
        <wlevs:source ref="helloworldAdapter"/>
    </wlevs:channel>
    <wlevs:channel id="helloworldOutstream" advertise="true">
        <wlevs:listener>
            <bean class="com.bea.wlevs.example.helloworld.HelloWorldBean"/>
        </wlevs:listener>
        <wlevs:source ref="helloworldProcessor"/>
    </wlevs:channel>
</beans>
```

D.2 wlevs:adapter

Use this element to declare an adapter component to the Spring application context.

D.2.1 Child Elements

The `wlevs:adapter` application assembly element supports the following child elements:

- `wlevs:listener`

- `wlevs:instance-property`

- `wlevs:property`

D.2.2 Attributes

Table D–1 lists the attributes of the `wlevs:adapter` application assembly element.

Table D–1   Attributes of the wlevs:adapter Application Assembly Element

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 D–1   (Cont.) Attributes of the wlevs:adapter Application Assembly Element

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
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

Table D–1 (Cont.) Attributes of the wlevs:adapter Application Assembly Element

Attribute	Description	Data Type	Required?
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

D.2.3 Example

The following example shows how to use the `wlevs:adapter` element 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.

D.3 wlevs:application-timestamped

Use this element to specify if an `wlevs:channel` is application timestamped, that is, if the application is responsible for assigning a timestamp to each event, using any time domain.

Otherwise, `wlevs:channel` is system timestamped, that is, the Oracle CEP server is responsible for assigning a timestamp to each event using `System.nanoTime()`.

D.3.1 Child Elements

The `wlevs:application-timestamped` application assembly element supports the following child elements.

- `wlevs:expression`—Specifies an expression to be used as an application timestamp for event processing.

D.3.2 Attributes

Table D–2 lists the attributes of the `wlevs:application-timestamped` application assembly element.

Table D–2 Attributes of the wlevs:application-timestamped Application Assembly Element

Attribute	Description	Data Type	Required?
is-total-order	Indicates if the application time published is always strictly greater than the last value used. Valid values are true or false. Default: false. For more information, see "Time" in the *Oracle CEP CQL Language Reference*.	Boolean	No.

D.3.3 Example

The following example shows how to use the `wlevs:application-timestamped` element in the EPN assembly file to specify an implicitly application timestamped channel:

```
<wlevs:channel id="fxMarketAmerOut" >
    <wlevs:application-timestamped>
    </wlevs:application-timestamped>
</wlevs:channel>
```

In the example, the application handles event timestamps internally.

The following example shows how to use `wlevs:application-timestamped` element in the EPN assembly file to specify an explicitly application timestamped channel by specifying the `wlevs:expression` element.

```
<wlevs:channel id="fxMarketAmerOut" >
    <wlevs:application-timestamped>
        <wlevs:expression>mytime+10</wlevs:expression>
    </wlevs:application-timestamped>
</wlevs:channel>
```

In the example, the `wlevs:expression` element defines the arithmetic expression used to assign a timestamp to each event.

D.4 wlevs:cache

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

D.4.1 Child Elements

The `wlevs:cache` application assembly element supports the following child elements.

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

  > **Note:** This child element is different from the `wlevs:caching-system` element used to *declare* a caching system. The child element of the `wlevs:cache` element 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.

D.4.2 Attributes

Table D–3 lists the attributes of the `wlevs:cache` application assembly element.

Table D–3 Attributes of the wlevs:cache Application Assembly Element

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.
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 Oracle CQL or 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.

D.4.3 Example

The following example shows how to use the `wlevs:cache` element 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`.

D.5 wlevs:cache-listener

Use this element 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 element is always a child of `wlevs:cache`.

D.5.1 Attributes

Table D–4 lists the attributes of the `wlevs:cache-listener` application assembly element.

Table D–4   Attributes of the wlevs:cache-listener Application Assembly Element

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.

D.5.2 Example

The following example shows how to use the `wlevs:cache-listener` element 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.

D.6 wlevs:cache-loader

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

This element is always a child of `wlevs:cache`.

D.6.1 Attributes

Table D–5 lists the attributes of the `wlevs:cache-loader` application assembly element.

Table D–5   Attributes of the wlevs:cache-loader Application Assembly Element

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.

D.6.2 Example

The following example shows how to use the `wlevs:cache-loader` element 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.

D.7 wlevs:cache-source

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

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

This element is a child of only `wlevs:processor` element.

D.7.1 Attributes

Table D–6 lists the attributes of the `wlevs:cache-source` application assembly
element.

Table D–6   Attributes of the wlevs:cache-source Application Assembly Element

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.

D.7.2 Example

The following example shows how to use the `wlevs:cache-source` element 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:channel 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` channel
as usual; however, the Oracle CQL or 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.

D.8 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 element is always a child of `wlevs:cache`.

D.8.1 Attributes

Table D–7 lists the attributes of the `wlevs:cache-store` application assembly element.

Table D–7   Attributes of the wlevs:cache-store Application Assembly Element

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.

D.8.2 Example

The following example shows how to use the `wlevs:cache-store` element 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.

D.9 wlevs:caching-system

Specifies the caching system used by the application.

D.9.1 Child Elements

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

- `wlevs:instance-property`
- `wlevs:property`

D.9.2 Attributes

Table D–8 lists the attributes of the `wlevs:caching-system` application assembly element.

Table D–8   Attributes of the wlevs:caching-system Application Assembly Element

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: `<wlevs:caching-system id="myCachingSystem" provider=coherence" />` Typically this attribute corresponds to the `provider-name` attribute of a `<factory>` Spring element 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

D.9.3 Example

The following example shows the simplest use of the `wlevs:caching-system` element 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.

D.10 wlevs:channel

Use this element to declare a channel to the Spring application context.

By default, channels assume that events are system timestamped. To configure application timestamped events, see child element `wlevs:application-timestamped`.

D.10.1 Child Elements

The `wlevs:channel` application assembly element supports the following child elements:

- `wlevs:listener`

- `wlevs:source`

- `wlevs:instance-property`

- `wlevs:property`

- `wlevs:application-timestamped`

D.10.2 Attributes

Table D–9 lists the attributes of the `wlevs:channel` application assembly element.

Table D–9 Attributes of the wlevs:channel Application Assembly Element

Attribute	Description	Data Type	Required?
id	Unique identifier for this component.	String	Yes.
	This identifier must correspond to the `<name>` element in the XML configuration file for this channel, if one exists.		
advertise	Advertises this service in the OSGi registry.	Boolean	No.
	Valid values are `true` and `false`. Default value is `false`.		
listeners	Specifies the components that listen to this component. Separate multiple components using commas.	String	No.
	Set this attribute to the value of the `id` attribute of the element (`wlevs:adapter`, `wlevs:channel`, or `wlevs:processor`) that defines the listening component.		
provider	Specifies the streaming provider.	String	No.
	Valid values are:		
	■ `oracle.channel`		
	Default value is `oracle.channel`, which is the out-of-the-box streaming provider.		
max-threads	Specifies the maximum number of threads that will be used to process events for this channel.	integer	No.
	When set to 0, the channel acts as a pass-through. When `max-threads > 0`, the channel acts as classic blocking queue, where upstream components are producers of events and the downstream components are the consumers of events. The queue size is defined by the configuration `max-size`. There will be up to `max-threads` number of threads consuming events from the queue.		
	You can change `max-threads` from 0 to a positive integer (that is, from a pass through to multiple threads) without redeploying. However, if you change `max-threads` from a positive integer to 0 (that is, from multiple threads to a pass through), then you must redeploy your application.		
	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.		
max-size	Specifies the maximum size of this channel.	integer	No.
	Zero-size streams synchronouslypass-through events. Streams with non-zero size process events asynchronously, buffering events by the requested size. If `max-threads` is zero, then `max-size` is zero.		
	The default value for this attribute is `1024`.		

Table D–9   (Cont.) Attributes of the wlevs:channel Application Assembly Element

Attribute	Description	Data Type	Required?
is-relation	Specifies the kind of events that are allowed to pass through the event channel. Two kind of events are supported: streams and relations. Streams are append-only. Relations support insert, delete, and updates. The default value for this attribute is `false`.	Boolean	No.
source	Specifies the component from which the channel sources events. Set this attribute to the value of the `id` attribute of the element (`wlevs:adapter`, `wlevs:channel`, or `wlevs:processor`) that defines the source component.	String	No.

D.10.3 Example

The following example shows how to use the `wlevs:channel` element in the EPN assembly file:

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

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

D.11 wlevs:event-bean

Use this element 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.

D.11.1 Child Elements

The `wlevs:event-bean` application assembly element supports the following child elements:

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

D.11.2 Attributes

Table D–10 lists the attributes of the `wlevs:event-bean` application assembly element.

Table D–10   Attributes of the wlevs:event-bean Application Assembly Element

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
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

Table D–10   (Cont.) Attributes of the wlevs:event-bean Application Assembly Element

Attribute	Description	Data Type	Required?
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

D.11.3 Example

The following example shows how to use the wlevs:event-bean element 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.

D.12 wlevs:event-type-repository

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

This element does not have any attributes.

D.12.1 Child Elements

The wlevs:event-type-repository application assembly element supports the following child element:

- wlevs:event-type

D.12.2 Example

The following example shows how to use the wlevs:event-type-repository element 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 element groups a single wlevs:event-type element to declare a single event type: HelloWorldEvent. See Section D.13, "wlevs:event-type" for additional details.

D.13 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 Oracle CQL and 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 element.

- Use the `wlevs:metadata` child element 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 element 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 element because you can them reuse the specified JavaBean class, and you control exactly what the event type looks like.

D.13.1 Child Elements

The `wlevs:event-type` application assembly element supports the following child elements:

- `wlevs:metadata`

- `wlevs:property`

D.13.2 Attributes

Table D–11 lists the attributes of the `wlevs:event-type` application assembly element.

Table D–11   Attributes of the wlevs:event-type Application Assembly Element

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.	String	No.
`type-name`	Specifies the name of this event type. This is the name you use whenever you reference the event type in the adapter, business POJO, or Oracle CQL or EPL rules.	String	Yes.

D.13.3 Example

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

```
<wlevs:event-type-repository>
        <wlevs:event-type type-name="SimpleEvent">
            <wlevs:properties>
                <wlevs:property name="msg" type="char" />
                <wlevs:property name="count" type="long" />
                <wlevs:property name="time_stamp" type="timestamp" />
        </wlevs:properties>
    </wlevs:event-type>
...
```

```
</wlevs:event-type-repository>
```

In the example, the name of the event type is `SimpleEvent` and its definition is determined by the `wlevs:property` elements. The values for the type attribute must conform to the `com.bea.wlevs.ede.api.Type` class.

For more information, see Section 1.1.3, "Event Type Data Types".

D.14 wlevs:expression

Use this element to specify an arithmetic expression in `wlevs:application-timestamped` to be used as an application timestamp for event processing.

For more information, see "arith_expr" in the *Oracle CEP CQL Language Reference*.

D.14.1 Example

The following example shows how to use `wlevs:expression` element in the EPN assembly file to specify an explicitly application timestamped channel.

```
<wlevs:channel id="fxMarketAmerOut" >
    <wlevs:application-timestamped>
        <wlevs:expression>mytime + 10</wlevs:expression>
    </wlevs:application-timestamped>
</wlevs:channel>
```

In the example, the `wlevs:expression` element defines the arithmetic expression used to assign a timestamp to each event.

D.15 wlevs:factory

Use this element to register a factory class as a service. Use of this element 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 element does not allow you to specify service properties. If you need to specify service properties, then you must use the Spring- OSGi `osgi:service` element instead.

This element does not have any child elements.

D.15.1 Attributes

Table D–12 lists the attributes of the `wlevs:factory` application assembly element.

Table D–12   Attributes of the wlevs:factory Application Assembly Element

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.

D.15.2 Example

The following example shows how to use the `wlevs:factory` element 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`.

D.16 wlevs:function

Use this element to specify a bean that contains user-defined functions for a processor. Oracle CEP supports both single-row and aggregate functions.

This element always has a standard Spring `bean` element either as a child or as a reference that specifies the Spring `bean` that implements the user-defined function.

For a single-row function for an Oracle CQL processor, you may specify one method on the implementing class as the function using the `exec-method` attribute. In this case, the method must be public and must be uniquely identifiable by its name (that is, the method cannot have been overridden). You may define an alias for the `exec-method` name using the `function-name` attribute. In the Oracle CQL query, you may call only the `exec-method` (either by its name or the `function-name` alias).

For a single-row function on an EPL processor, you may define an alias for the implementing class name using the `function-name` attribute. The `exec-method` name is not applicable in this case. In the EPL query, you may call any public or static method on the implementing class using either the implementing class name or the `function-name` alias.

For an aggregate function on either an Oracle CQL or EPL processor, the Spring `bean` must implement the following interfaces from the com.bea.wlevs.processor package:

- `AggregationFunctionFactory`
- `AggregationFunction`

For an aggregate function, the `exec-method` attribute is not applicable on both an Oracle CQL processor and an EPL processor.

For more information, see:

- "Functions: User-Defined" in the *Oracle CEP CQL Language Reference*
- "EPL Reference: Functions" in the *Oracle CEP EPL Language Reference*

D.16.1 Attributes

Table D–13 lists the attributes of the `wlevs:function` application assembly element.

Table D–13   Attributes of the wlevs:function Application Assembly Element

Attribute	Description	Data Type	Required?
exec-method	For a user-defined single-row function on an Oracle CQL processor, this element specifies the method name of the Spring `bean` that implements the function. For a user-defined single-row or aggregate function on an EPL processor or a user-defined aggregate function on an Oracle CQL processor, this attribute is not applicable.	String	No.
function-name	For a user-defined single-row function on an Oracle CQL processor, use this attribute to define an alias for the `exec-method` name. You can then use the `function-name` in your Oracle CQL query instead of the `exec-name`. For a user-defined single-row function on an EPL processor, use this attribute to define an alias for the implementing Spring `bean` class name. You can then use the `function-name` in your EPL query instead of the Spring `bean` class name and still invoke any public or static method that the Spring `bean` class implements. For a user-defined aggregate function on an Oracle CQL or EPL processor, use this attribute to define an alias for the implementing Spring `bean` class name. You can then use the `function-name` in your EPL query instead of the Spring `bean` class name. The default value is the Spring bean name.	String	No.
ref	Specifies the Spring `bean` that implements the function. Set this attribute to the value of the `id` attribute of the Spring `bean`. This is an alternative to making the Spring `bean` element a child of the `wlevs:function` element.	String	No.

D.16.2 Example

The following examples show how to use the `wlevs:function` element and its attributes on both Oracle CQL and EPL processors:

D.16.2.1 Single-Row User-Defined Function on an Oracle CQL Processor

Example D–1 shows how you implement a single-row user-defined the function for an Oracle CQL processor.

Example D–1   Single-Row User Defined Function Implementation Class

```
package com.bea.wlevs.example.function;

public class MyMod {
    public Object execute(Object[] args) {
        int arg0 = ((Integer)args[0]).intValue();
        int arg1 = ((Integer)args[1]).intValue();
        return new Integer(arg0 % arg1);
    }
}
```

Example D–2 shows how to use the wlevs:function to define a single-row function on an Oracle CQL processor in the EPN assembly file.

Example D–2 Single-Row User Defined Function for an Oracle CQL Processor

```
<wlevs:processor id="testProcessor">
    <wlevs:listener ref="providerCache"/>
    <wlevs:listener ref="outputCache"/>
    <wlevs:cache-source ref="testCache"/>
    <wlevs:function function-name="mymod" exec-method="execute" />
        <bean class="com.bea.wlevs.example.function.MyMod"/>
    </wlevs:function>
</wlevs:processor>
```

Example D–3 shows how you invoke the function in an Oracle CQL query.

Example D–3 Invoking the Single-Row User-Defined Function on an Oracle CQL Processor

```
...
<view id="v1" schema="c1 c2 c3 c4"><![CDATA[
    select
        mymod(c1, 100), c2, c3, c4
    from
        S1
]]></view>
...
<query id="q1"><![CDATA[
    select * from v1 [partition by c1 rows 1] where c4 - c3 = 2.3
]]></query>
...
```

D.16.2.2 Single-Row User-Defined Function on an EPL Processor

Example D–4 shows how you implement a single-row user-defined the function for an EPL processor.

Example D–4 Single-Row User Defined Function Implementation Class

```
package com.bea.wlevs.example.function;

public class LegacyMathBean {
    public Object square(Object[] args) {
    ...
    }
    public Object cube(Object[] args) {
    ...
    }
}
```

Example D–5 shows how to use the wlevs:function to define a single-row function for an EPL processor in the EPN assembly file.

Example D–5 Single-Row User Defined Function for an EPL Processor

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

Example D–6 shows how you invoke the function in an EPL query.

Example D–6   Invoking the Single-Row User-Defined Function on an EPL Processor

```
<rule><![CDATA[
    select math.square(inputValue) ...
]]></rule>
```

D.16.2.3 Aggregate User-Defined Function on an Oracle CQL Processor

Example D–7 shows how to implement a user-defined aggregate function for an Oracle CQL processor.

Example D–7   Aggregate User Defined Function Implementation Class

```
package com.bea.wlevs.test.functions;

import com.bea.wlevs.processor.AggregationFunction;
import com.bea.wlevs.processor.AggregationFunctionFactory;

public class Variance implements AggregationFunctionFactory, AggregationFunction {

    private int count;
    private float sum;
    private float sumSquare;

    public Class<?>[] getArgumentTypes() {
        return new Class<?>[] {Integer.class};
    }

    public Class<?> getReturnType() {
        return Float.class;
    }

    public AggregationFunction newAggregationFunction() {
        return new Variance();
    }

    public void releaseAggregationFunction(AggregationFunction function) {
    }

    public Object handleMinus(Object[] params) {
        if (params != null && params.length == 1) {
            Integer param = (Integer) params[0];
            count--;
            sum -= param;
            sumSquare -= (param * param);
        }

        if (count == 0) {
            return null;
        } else {
            return getVariance();
        }
    }

    public Object handlePlus(Object[] params) {
        if (params != null && params.length == 1) {
            Integer param = (Integer) params[0];
            count++;
            sum += param;
            sumSquare += (param * param);
        }
```

```
        if (count == 0) {
            return null;
        } else {
            return getVariance();
        }
    }

    public Float getVariance() {
        float avg = sum / (float) count;
        float avgSqr = avg * avg;
        float var = sumSquare / (float)count - avgSqr;
        return var;
    }

    public void initialize() {
        count = 0;
        sum = 0.0F;
        sumSquare = 0.0F;
    }

}
```

shows how to use the `wlevs:function` to define an aggregate function on an Oracle CQL processor in the EPN assembly file.

Example D–8   Aggregate User Defined Function for an Oracle CQL Processor

```
  <wlevs:processor id="testProcessor">
     <wlevs:listener ref="providerCache"/>
     <wlevs:listener ref="outputCache"/>
     <wlevs:cache-source ref="testCache"/>
     <wlevs:function function-name="var">
       <bean class="com.bea.wlevs.test.functions.Variance"/>
     </wlevs:function>
   </wlevs:processor>
```

shows how you invoke the function in an Oracle CQL query.

Example D–9   Invoking the Aggregate User-Defined Function on an Oracle CQL Processor

```
...
<query id="uda6"><![CDATA[
    select var(c2) from S4[range 3]
]]></query>
...
```

D.16.2.4 Aggregate User-Defined Function on an EPL Processor

shows how to implement a user-defined aggregate function for an EPL processor.

Example D–10   Aggregate User Defined Function Implementation Class

```
package com.bea.wlevs.test.functions;

import com.bea.wlevs.processor.AggregationFunction;
import com.bea.wlevs.processor.AggregationFunctionFactory;

public class Variance implements AggregationFunctionFactory, AggregationFunction {

    private int count;
    private float sum;
```

```
private float sumSquare;

public Class<?>[] getArgumentTypes() {
    return new Class<?>[] {Integer.class};
}

public Class<?> getReturnType() {
    return Float.class;
}

public AggregationFunction newAggregationFunction() {
    return new Variance();
}

public void releaseAggregationFunction(AggregationFunction function) {
}

public Object handleMinus(Object[] params) {
    if (params != null && params.length == 1) {
        Integer param = (Integer) params[0];
        count--;
        sum -= param;
        sumSquare -= (param * param);
    }

    if (count == 0) {
        return null;
    } else {
        return getVariance();
    }
}

public Object handlePlus(Object[] params) {
    if (params != null && params.length == 1) {
        Integer param = (Integer) params[0];
        count++;
        sum += param;
        sumSquare += (param * param);
    }

    if (count == 0) {
        return null;
    } else {
        return getVariance();
    }
}

public Float getVariance() {
    float avg = sum / (float) count;
    float avgSqr = avg * avg;
    float var = sumSquare / (float)count - avgSqr;
    return var;
}

public void initialize() {
    count = 0;
    sum = 0.0F;
    sumSquare = 0.0F;
}

}
```

Example D–11 shows how to use the `wlevs:function` to define an aggregate function on an EPL processor in the EPN assembly file.

Example D–11 Aggregate User Defined Function for an EPL Processor

```
<wlevs:processor id="testProcessor" provider="epl">
   <wlevs:listener ref="providerCache"/>
   <wlevs:listener ref="outputCache"/>
   <wlevs:cache-source ref="testCache"/>
   <wlevs:function function-name="var">
     <bean class="com.bea.wlevs.test.functions.Variance"/>
   </wlevs:function>
 </wlevs:processor>
```

Example D–12 shows how you invoke the function in an EPL query.

Example D–12 Invoking the Aggregate User-Defined Function on an EPL Processor

```
...
<rule><![CDATA[
    select var(c2) from S4
]]></rule>
...
```

D.16.2.5 Specifying the Implementation Class: Nested Bean or Reference

Example D–13 shows how to use the `wlevs:function` element with a nested `bean` element in the EPN assembly file.

Example D–13 User Defined Function Using Nested Bean Element

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

Example D–14 shows how to use the `wlevs:function` element and its `ref` attribute to reference a `bean` element defined outside of the `wlevs:function` element in the EPN assembly file.

Example D–14 User Defined Function Using Reference

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

D.17 wlevs:instance-property

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

This element is used only as a child of `wlevs:adapter`, `wlevs:processor`, `wlevs:channel`, or `wlevs:caching-system`.

The `wlevs:instance-property` element is defined as the Spring `propertyType` type; for additional details of this Spring data type, the definition of the allowed child elements, and so on, see the http://www.springframework.org/schema/beans/spring-beans-2.0.xsd .

D.17.1 Child Elements

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

- `meta`

- `bean`

- `ref`

- `idref`

- `value`

- `null`

- `list`

- `set`

- `map`

- `props`

D.17.2 Attributes

Table D–14 lists the attributes of the `wlevs:instance-property` application assembly element.

Table D–14   Attributes of the wlevs:instance-property Application Assembly Element

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.

D.17.3 Example

The following example shows how to use the `wlevs:instance-property` element 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` element above sets the value of this property to `HelloWorld - the current time is:`.

D.18 wlevs:listener

Specifies the component that listens to the component to which this element 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.

> **Caution:** Nested definitions are not eligible for dynamic configuration or monitoring.

This element is always a child of `wlevs:adapter`, `wlevs:processor`, `wlevs:channel`, or `wlevs:caching-system`.

D.18.1 Attributes

Table D–15 lists the attributes of the `wlevs:listener` application assembly element.

Table D–15   Attributes of the wlevs:listener Application Assembly Element

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.

D.18.2 Example

The following example shows how to use the `wlevs:listener` element 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:channel`, or `wlevs:processor` element whose unique identifier is `helloworldOutstream`.

D.19 wlevs:metadata

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

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

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

The `wlevs:metadata` element is defined as the Spring `mapType` type; for additional details of this Spring data type, see the http://www.springframework.org/schema/beans/spring-beans-2.0.xsd .

D.19.1 Child Elements

The `wlevs:metadata` element can have one or more standard Spring `entry` child elements as defined in the
http://www.springframework.org/schema/beans/spring-beans-2.0.xsd
.

D.19.2 Attributes

Table D–16 lists the attributes of the `wlevs:metadata` application assembly element.

Table D–16   Attributes of the wlevs:metadata Application Assembly Element

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

D.19.3 Example

The following example shows how to use the `wlevs:metadata` element 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` element groups together four standard Spring `entry` elements 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.

D.20 wlevs:processor

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

D.20.1 Child Elements

The `wlevs:processor` Spring element supports the following child elements:

- wlevs:instance-property

- wlevs:listener

- wlevs:property

- wlevs:cache-source

- wlevs:table-source

- wlevs:function

D.20.2 Attributes

Table D–17 lists the attributes of the `wlevs:processor` application assembly element.

Table D–17   Attributes of the wlevs:processor Application Assembly Element

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 Oracle CQL or 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 Oracle Continuous Query Language (Oracle CQL) or Event Processor Language (EPL). Valid values are: ■ `epl` ■ `cql` The default value is `cql`.	String	No.
queryURL	Specifies a URL that points to an Oracle CQL or EPL rules definition file for this processor.	String.	No.

D.20.3 Example

The following example shows how to use the `wlevs:processor` element 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 Oracle CQL rules for this processor, the `name` element must contain the value `spreader`. This way Oracle CEP knows which Oracle CQL or EPL rules it must file for this particular processor.

D.21 wlevs:property

Specifies a custom property to apply to the event type.

This element is used only as a child of `wlevs:adapter`, `wlevs:event-type`, `wlevs:processor`, `wlevs:channel`, or `wlevs:caching-system`.

The `wlevs:property` element is defined as the Spring `propertyType` type; for additional details of this Spring data type, the definition of the allowed child elements, and so on, see the http://www.springframework.org/schema/beans/spring-beans-2.0.xsd .

D.21.1 Child Elements

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

- meta

- bean

- ref

- idref

- value

- null

- list

- set

- map

- props

D.21.2 Attributes

Table D–18 lists the attributes of the `wlevs:property` application assembly element.

Table D–18   Attributes of the wlevs:property Application Assembly Element

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.

D.21.3 Example

The following example shows how to use the `wlevs:property` element 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` element defines a custom property of the `ForeignExchangeEvent` called `builderFactory`. The property uses the standard Spring `bean` element to specify the Spring bean used as a factory to create `ForeignExchangeEvents`.

D.22 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.

> **Caution:** Nested definitions are not eligible for dynamic
> configuration or monitoring.

This element is a child of `wlevs:channel` or `wlevs:processor`.

D.22.1 Attributes

Table D–19 lists the attributes of the `wlevs:source` application assembly element.

Table D–19   Attributes of the wlevs:source Application Assembly Element

Attribute	Description	Data Type	Required?
`ref`	Specifies the source of the channel to which this element 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.

D.22.2 Example

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

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

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

D.23 wlevs:table

Specifies a relational database table that supplies data to one or more processor components. The processor components in turn are associated with an Oracle CQL query that directly references the table.

D.23.1 Attributes

Table D–20 lists the attributes of the `wlevs:table` application assembly element.

Table D–20   Attributes of the wlevs:table Application Assembly Element

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 table.	`String`	Yes.
`event-type`	The name of the event type associated with this table as defined in the event type repository.	`String`	Yes.
`data-source`	The name of the relational data source defined in the Oracle CEP server configuration file used to access this database table.	`String`	Yes.

D.23.2 Example

The following example shows how to use the `wlevs:table` element in the EPN assembly file:

```
<wlevs:table id="Stock" event-type="StockEvent" data-source="StockDs" />

<wlevs:processor id="proc">
    <wlevs:table-source ref="Stock" />
</wlevs:processor>
```

In this example, a `wlevs:processor` references the table using its `wlevs:table-source` element.

D.24 wlevs:table-source

Specifies a relational database table that supplies data to this processor component. The processor component in turn is associated with an Oracle CQL query that directly references the table.

This element is a child of only `wlevs:processor` element.

D.24.1 Attributes

Table D–21 lists the attributes of the `wlevs:table-source` application assembly element.

Table D–21   Attributes of the wlevs:table-source Application Assembly Element

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

D.24.2 Example

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

```
<wlevs:table id="Stock" event-type="StockEvent" data-source="StockDs" />

<wlevs:processor id="proc">
    <wlevs:table-source ref="Stock" />
</wlevs:processor>
```

E

Schema Reference: Deployment deployment.xsd

This appendix describes the elements of the `deployment.xsd` schema.

For more information, see:

- Section E.1, "Overview of the Oracle CEP Deployment Elements"
- Section B.3, "Deployment Schema deployment.xsd"

E.1 Overview of the Oracle CEP Deployment Elements

Oracle CEP provides a number of application assembly elements 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.

E.1.1 Element Hierarchy

The Oracle CEP component configuration elements are organized into the following hierarchy:

beans
    Standard Spring and OSGi elements such as bean, osgi-service, and so on.

E.1.2 Example of an Oracle CEP Deployment Configuration File

The following sample deployment configuration file from the fx application shows how to use many of the Oracle CEP elements:

```
<?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_11.1.0.0.jar"/>
</beans>
```

E.2 wlevs:deployment

Use this element to declare an adapter component to the Spring application context.

E.2.1 Child Elements

The `wlevs:deployment` deployment element has no child elements:

E.2.2 Attributes

Table E–1 lists the attributes of the `wlevs:deployment` deployment element.

Table E–1 Attributes of the wlevs:deployment Deployment Element

Attribute	Description	Data Type	Required?
id	Unique identifier for this deployed application.	String	Yes.
depends-on	The names of the beans that this deployment bean depends on being initialized. The bean factory will guarantee that these beans get initialized before this bean.	String	Yes.
location	URL that specifies the location of the bundle that is to be deployed.If a relative URL is specified then the location is relative the DOMAIN_DIR domain directory. For example: `location="file:applications/simpleApp/simpleApp.jar"` Specifies that the bundle simpleApp.jar, located in the *DOMAIN_DIR*/applications/simpleApp directory, is to be deployed to Oracle CEP server.	String	No.
state	Specifies the state that the bundle should be in once it is deployed to the Oracle CEP server. The value of this attribute must be one of the following: ■ `start`: Install and start the bundle so that it immediately begins taking client requests. ■ `install`: Install the bundle, but do not start it. ■ `update`: Update an existing bundle. Default value: `start`.	String	No.

E.2.3 Example

The following example shows how to use the `wlevs:deployment` element in the deployment file:

```xml
<?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_11.1.0.0.jar"/>
</beans>
```

F

Schema Reference: Server Configuration wlevs_server_config.xsd

This appendix describes the elements of the `welvs_server_config.xsd` schema.

For more information, see:

F.1 Overview of the Oracle CEP Server Configuration Elements

Oracle CEP provides a number of server configuration elements that you use to configure Oracle CEP server-specific attributes and services.

F.1.1 Element Hierarchy

The top-level Oracle CEP server configuration elements are organized into the following hierarchy:

- `config`
    - `domain`
    - `rmi`
    - `jndi-context`
    - `exported-jndi-context`
    - `jmx`
    - `transaction-manager`
    - `work-manager`
    - `logging-service`
    - `log-stdout`
    - `log-file`
    - `jetty-web-app`
    - `netio`
    - `jetty`
    - `netio-client`
    - `debug`

- data-source

- http-pubsub

- event-store

- cluster

- rdbms-event-store-provider

- user-event-store-provider

- ssl

- weblogic-rmi-client

- weblogic-jta-gateway

- use-secure-connections

- show-detail-error-message

- cql

F.1.2 Example of an Oracle CEP Server Configuration File

The following sample Oracle CEP server configuration file from the HelloWorld application shows how to use many of the Oracle CEP elements:

```xml
<?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">
  <domain>
     <name>WLEventServerDomain</name>
  </domain>

  <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/evsidentity.jks</key-store>
  <key-store-pass>
      <password>{Salted-3DES}s4YUEvH4Wl2DAjb45iJnrw==</password>
  </key-store-pass>
  <key-store-alias>evsidentity</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>
      <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>

<!-- Sample cluster configuration -->
<!--
<cluster>
  <server-name>myServer</server-name>
  <multicast-address>239.255.0.1</multicast-address>
  <enabled>coherence</enabled>
  <security>none</security>
  <groups></groups>
</cluster>
-->

<logging-service>
  <name>myLogService</name>
  <log-file-config>myFileConfig</log-file-config>
```

```
    <stdout-config>myStdoutConfig</stdout-config>
    <logger-severity>Notice</logger-severity>
    <!-- logger-severity-properties is used to selectively enable logging for
     individual categories -->
    <!--logger-severity-properties>
      <entry>
        <key>org.springframework.osgi.extender.internal.dependencies.startup</key>
        <value>Debug</value>
      </entry>
    </logger-severity-properties-->
  </logging-service>

  <log-file>
    <name>myFileConfig</name>
    <rotation-type>none</rotation-type>
  </log-file>

  <log-stdout>
    <name>myStdoutConfig</name>
    <stdout-severity>Debug</stdout-severity>
  </log-stdout>

</n1:config>
```

F.2 auth-constraint

Use this element to configure an authorization constraint for a
channel-constraints element.

For more information on channels, see channels.

F.2.1 Child Elements

The auth-constraint server configuration element supports the child elements that
Table F–1 lists

Table F–1   Child Elements of: auth-constraint

XML Tag	Type	Description
description	string	The description of the role.
role-name	string	A valid role name.
		"Users, Groups, and Roles" in the *Oracle CEP Administrator's Guide*

F.2.2 Attributes

The auth-constraint server configuration element has no attributes.

F.2.3 Example

The following example shows how to use the auth-constraint element in the
Oracle CEP server configuration file:

```
<http-pubsub>
     <name>myPubsub</name>
     <path>/pubsub</path>
     <pub-sub-bean>
...
         <channel-constraints>
             <element>
```

...

```
                <auth-constraint>
                    <description>Administrators</description>
                    <role-name>admin</role-name>
                </auth-constraint>
            </element>
        </channel-constraints>
    </pub-sub-bean>
</http-pubsub>
```

F.3 channels

Use this element to configure one or more channels for a pubsub-bean element.

Channel patterns always begin with a forward slash (/). Clients subscribe to these channels to either publish or receive messages

F.3.1 Child Elements

The `channels` server configuration element contains one or more `element` child elements that each contain a `channel-pattern` child element and zero or more `message-filters` child elements. Each `message-filters` child element contains an `element` child element with the `string` value of a `message-filter-name` that corresponds to a message-filters element.

F.3.2 Attributes

The `channels` server configuration element has no attributes.

F.3.3 Example

The following example shows how to use the `channels` element in the Oracle CEP server configuration file:

```
<http-pubsub>
    <name>myPubsub</name>
    <path>/pubsub</path>
    <pub-sub-bean>
        <server-config>
            <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>
```

F.4 channel-constraints

Use this element to configure one or more channel constraints for a `pubsub-bean` element.

For more information on channels, see `channels`.

F.4.1 Child Elements

The `channel-constraints` server configuration element contains one or more `element` child element that each support the following child elements:

- `channel-resource-collection`
- `auth-constraint`

F.4.2 Attributes

The `channel-constraints` server configuration element has no attributes.

F.4.3 Example

The following example shows how to use the `channel-constraints` element in the Oracle CEP server configuration file:

```
<http-pubsub>
    <name>myPubsub</name>
    <path>/pubsub</path>
    <pub-sub-bean>
...
        <channel-constraints>
            <element>
                <channel-resource-collection>
                    <element>
                        <channel-resource-name>Foo</channel-resource-name>
                        <descriptions>
                            <element>Foo</element>
                        </descriptions>
                        <channel-patterns>
                            <element>Foo</element>
                        </channel-patterns>
                        <channel-operations>
                            <element>Foo</element>
                        </channel-operations>
                    </element>
                </channel-resource-collection>
                <auth-constraint>
                    <description>Foo</description>
                    <role-name>Foo</role-name>
                </auth-constraint>
            </element>
        </channel-constraints>
    </pub-sub-bean>
</http-pubsub>
```

F.5 channel-resource-collection

Use this element to configure one or more channel resource collections for a
`channel-constraints` element.

For more information on channels, see `channels`.

F.5.1 Child Elements

The `channel-resource-collection` server configuration element contains zero or
more `element` child elements that support the child elements that Table F–2 lists

Table F–2   Child Elements of: channel-resource-collection

XML Tag	Type	Description
`channel-resource-name`	`string`	The name of this channel resource.
`descriptions`	`string`	Description of this channel resource collection. This element contains an `element` child element with a `string` value.
`channel-patterns`	`string`	Specifies a channel pattern. This element contains an `element` child element with a `string` value.
`channel-operations`	`string`	Specifies the operation to channel, validate values include: ■　`create` ■　`delete` ■　`subscribe` ■　`publish` This element contains an `element` child element with a `string` value.

F.5.2 Attributes

The `channel-resource-collection` server configuration element has no
attributes.

F.5.3 Example

The following example shows how to use the `channel-resource-collection`
element in the Oracle CEP server configuration file:

```
<http-pubsub>
    <name>myPubsub</name>
    <path>/pubsub</path>
    <pub-sub-bean>
...
        <channel-constraints>
            <element>
                <channel-resource-collection>
                    <element>
                        <channel-resource-name>Foo</channel-resource-name>
                        <descriptions>
                            <element>Foo</element>
                        </descriptions>
                        <channel-patterns>
                            <element>Foo</element>
                        </channel-patterns>
                        <channel-operations>
                            <element>Foo</element>
```

```
                        </channel-operations>
                    </element>
                </channel-resource-collection>
                <auth-constraint>
                    <description>Foo</description>
                    <role-name>Foo</role-name>
                </auth-constraint>
            </element>
        </channel-constraints>
    </pub-sub-bean>
</http-pubsub>
```

F.6 cluster

Use this element to configure a cluster component in the Oracle CEP server.

For more information, see "Administrating Oracle CEP Multi-Server Domains" in the *Oracle CEP Administrator's Guide*.

F.6.1 Child Elements

The `cluster` server configuration element supports the child elements that Table F–3 lists.

Table F–3   Child Elements of: cluster

XML Tag	Type	Description
name	string	The name of this cluster. For more information, see name.
server-name	string	Specifies a unique name for the server. Oracle CEP Visualizer uses the value of this element when it displays the server in its console. Default value: ■ Oracle CEP native clustering: `WLEvServer-identity` where `identity` is the value of the `identity` element. ■ Oracle Coherence: `WLEvServer-identity` where `identity` is the member ID as determined by Oracle Coherence.
server-host-name	string	Specifies the host address or IP used for point-to-point HTTP multi-server communication. Default value is the IP address associated with the default NIC for the machine.
multicast-address	string	This child element is required unless all servers of the multi-server domain are hosted on the same computer; in that case you can omit the `multicast-address` element and Oracle CEP automatically assigns a multicast address to the multi-server domain based on the computer's IP address. If, however, the servers are hosted on different computers, then you must provide an appropriate domain-local address. Oracle recommends you use an address of the form `239.255.X.X`, which is what the auto-assigned multicast address is based on. Using Oracle Coherence, there is also an extension: if you use a unicast address then Oracle Coherence will be configured in Well Known Address (WKA) mode. This is necessary in environments that do not support mulitcast.

Table F–3   (Cont.) Child Elements of: cluster

XML Tag	Type	Description
multicast-interface	string	The name of the interface that the multicast address should be bound to. This can be one of: - Simple name, such as eth0. - IP address to which the NIC is bound, such as 192.168.1.2. - IP address and network mask to which the NIC is bound separated by a /, such as 192.68.1.2/255.255.255.0.
multicast-port	int	Specifies the port used for multicast traffic. Default value is 9100.
identity	string	Applicable only to Oracle CEP native clustering: specifies the server's identity and must be an integer between 1 and INT_MAX. Oracle CEP numerically compares the server identities during multi-server operations; the server with the lowest identity becomes the domain coordinator. Be sure that each server in the multi-server domain has a different identity; if servers have the same identity, the results of multi-server operations are unpredictable. Not applicable to Oracle Coherence.
enabled	See Description	Specifies whether or not the cluster is enabled. Valid values: - coherence - evs4j - true: cluster is enabled (Oracle Coherence mode) - false: cluster is not enabled (default).
security	See Description	Specifies the type of security for this cluster. Valid values: - none—Default value. Specifies that no security is configured for the multi-server domain. - encrypt—Specifies that multi-server messages should be encrypted.
groups	string	Specifies a comma-separated list of the names of the groups this cluster belongs to. For more information, see "Groups" in the *Oracle CEP Administrator's Guide*.
operation-timeout	int	Specifies, in milliseconds, the timeout for point-to-point HTTP multi-server requests. Default value is 30000.

F.6.2 Attributes

The cluster server configuration element has no attributes.

F.6.3 Example

The following example shows how to use the cluster element in the Oracle CEP server configuration file:

```
<cluster>
    <name>MyCluster</name>
    <server-name>myServer1</server-name>
    <multicast-address>239.255.0.1</multicast-address>
    <identity>1</identity>
    <enabled>true</enabled>
</cluster>
```

In the example, the cluster element's unique identifier is MyCluster.

F.7 connection-pool-params

Use this element to specify connection pool-related `data-source` parameters.

F.7.1 Child Elements

The `connection-pool-params` server configuration element supports the child elements that Table F–4 lists.

Table F–4   Child Elements of: connection-pool-params

XML Tag	Type	Description
statement-time out	int	The time after which a statement currently being executed will time out. `statement-timeout` relies on underlying JDBC driver support. The server passes the time specified to the JDBC driver using the `java.sql.Statement.setQueryTimeout()` method. If your JDBC driver does not support this method, it may throw an exception and the timeout value is ignored. A value of -1 disables this feature. A value of 0 means that statements will not time out. Default: -1.
profile-harves t-frequency-se conds	int	The number of seconds between diagnostic profile harvest operations. Default: 300.
inactive-conne ction-timeout- seconds	int	The number of inactive seconds on a reserved connection before the connection is reclaimed and released back into the connection pool. Default: 0.
shrink-frequen cy-seconds	int	The number of seconds to wait before shrinking a connection pool that has incrementally increased to meet demand. Default: 900.
driver-interce ptor	string	Specifies the absolute name of the application class used to intercept method calls to the JDBC driver. The application specified must implement the `weblogic.jdbc.extensions.DriverInterceptor` interface.
seconds-to-tru st-an-idle-poo l-connection	int	The number of seconds within a connection use that the server trusts that the connection is still viable and will skip the connection test, either before delivering it to an application or during the periodic connection testing process. Default: 10.
pinned-to-thre ad	boolean	This option can improve performance by enabling execute threads to keep a pooled database connection even after the application closes the logical connection. Default: `false`.
test-connectio ns-on-reserve	boolean	Test a connection before giving it to a client. Requires that you specify `test-table-name`. Default: `false`.
profile-type	int	Specifies that type of profile data to be collected.
statement-cach e-type	string	The algorithm used for maintaining the prepared statements stored in the statement cache. Valid values: ■ `LRU` - when a new prepared or callable statement is used, the least recently used statement is replaced in the cache ■ `FIXED` - the first fixed number of prepared and callable statements are cached Default: `LRU`.
connection-res erve-timeout-s econds	int	The number of seconds after which a call to reserve a connection from the connection pool will timeout. When set to 0, a call will never timeout. When set to -1, a call will timeout immediately. Default: -1.

Table F–4   (Cont.) Child Elements of: connection-pool-params

XML Tag	Type	Description
credential-mapping-enabled	boolean	Enables the server to set a light-weight client ID on the database connection based on a map of database IDs when an application requests a database connection. Default: `false`.
login-delay-seconds	int	The number of seconds to delay before creating each physical database connection. This delay supports database servers that cannot handle multiple connection requests in rapid succession. The delay takes place both during initial data source creation and during the lifetime of the data source whenever a physical database connection is created. Default: 0.
test-table-name	string	The name of the database table to use when testing physical database connections. This name is required when you specify `test-frequency-seconds` and enable `test-reserved-connections`. The default SQL code used to test a connection is `select count(*) from` *test-table-name* where *test-table-name* is the value of the test-table-name element. Most database servers optimize this SQL to avoid a table scan, but it is still a good idea to set `test-table-name` to the name of a table that is known to have few rows, or even no rows. If test-table-name begins with `SQL`, then the rest of then the rest of the string following that leading token will be taken as a literal SQL statement that will be used to test connections instead of the standard query.
statement-cache-size	int	The number of prepared and callable statements stored in the cache between 1 and 1024. This may increase server performance. Default: 10.
init-sql	string	SQL statement to execute that will initialize newly created physical database connections. Start the statement with `SQL` followed by a space.
connection-creation-retry-frequency-seconds	int	The number of seconds between attempts to establish connections to the database. If you do not set this value, data source creation fails if the database is unavailable. If set and if the database is unavailable when the data source is created, the server will attempt to create connections in the pool again after the number of seconds you specify, and will continue to attempt to create the connections until it succeeds. When set to 0, connection retry is disabled. Default: 0.
test-frequency-seconds	int	The number of seconds between when the server tests unused connections. (Requires that you specify a Test Table Name.) Connections that fail the test are closed and reopened to re-establish a valid physical connection. If the test fails again, the connection is closed. In the context of multi data sources, this attribute controls the frequency at which the server checks the health of data sources it had previously marked as unhealthy. When set to 0, the feature is disabled. Default: 120.
jdbc-xa-debug-level	int	Specifies the JDBC debug level for XA drivers. Default: 10.
initial-capacity	int	The number of physical connections to create when creating the connection pool in the data source. If unable to create this number of connections, creation of the data source will fail. Default: 1.
max-capacity	int	The maximum number of physical connections that this connection pool can contain. Default: 15.
capacity-increment	int	The number of connections created when new connections are added to the connection pool. Default: 1.

Table F–4   (Cont.) Child Elements of: connection-pool-params

XML Tag	Type	Description
highest-num-wa iters	int	The maximum number of connection requests that can concurrently block threads while waiting to reserve a connection from the data source's connection pool. Default: Integer.MAX_VALUE.

F.7.2 Attributes

The connection-pool-params server configuration element has no attributes.

F.7.3 Example

The following example shows how to use the connection-pool-params element in the Oracle CEP server configuration file:

```
<data-source>
    <name>orads</name>
    <xa-params>
        <keep-xa-conn-till-tx-complete>true</keep-xa-conn-till-tx-complete>
    </xa-params>
    <driver-params>
        <url>jdbc:oracle:thin:@localhost:1521:ce102</url>
        <driver-name>oracle.jdbc.OracleDriver</driver-name>
        <properties>
            <element>
                <name>user</name>
                <value>wlevs</value>
            </element>
            <element>
                <name>password</name>
                <value>wlevs</value>
            </element>
        </properties>
    </driver-params>
    <connection-pool-params>
        <initial-capacity>5</initial-capacity>
        <max-capacity>10</max-capacity>
        <test-table-name>SQL SELECT 1 FROM DUAL</test-table-name>
        <test-frequency-seconds>5</test-frequency-seconds>
    </connection-pool-params>
    <data-source-params>
        <jndi-names>
            <element>orads</element>
        </jndi-names>
        <global-transactions-protocol>None</global-transactions-protocol>
    </data-source-params>
</data-source>
```

F.8 cql

Use this element to configure Oracle CQL-specific options in the Oracle CEP server.

F.8.1 Child Elements

The cql server configuration element supports the following child elements:

- name
- scheduler

F.8.2 Attributes

The `cql` server configuration element has no attributes.

F.8.3 Example

The following example shows how to use the `cql` element in the Oracle CEP server configuration file:

```
<cql>
    <name>myCQL</name>
    <storage>
        <folder>myfolder</folder>
        <metadata-name>myname</metadata-name>
    </storage>
    <scheduler>
        <class-name>myclass</class-name>
        <threads>10</threads>
        <direct-interop>false</direct-interop>
    </scheduler>
</cql>
```

In the example, the `cql` element's unique identifier is `myCQL`.

F.9 data-source

This configuration type defines configuration for a DataSource service.

F.9.1 Child Elements

The `data-source` server configuration element supports the following child elements:

- name

- xa-params

- data-source-params

- connection-pool-params

- driver-params

F.9.2 Attributes

The `data-source` server configuration element has no attributes.

F.9.3 Example

The following example shows how to use the `data-source` element in the Oracle CEP server configuration file:

```
<data-source>
    <name>orads</name>
    <driver-params>
        <url>jdbc:oracle:thin:@localhost:1521:ce102</url>
        <driver-name>oracle.jdbc.OracleDriver</driver-name>
        <properties>
            <element>
                <name>user</name>
                <value>wlevs</value>
            </element>
```

```
                    <element>
                        <name>password</name>
                        <value>wlevs</value>
                    </element>
            </properties>
        </driver-params>
        <connection-pool-params>
            <initial-capacity>5</initial-capacity>
            <max-capacity>10</max-capacity>
            <test-table-name>SQL SELECT 1 FROM DUAL</test-table-name>
            <test-frequency-seconds>5</test-frequency-seconds>
        </connection-pool-params>
        <data-source-params>
            <jndi-names>
                <element>orads</element>
            </jndi-names>
            <global-transactions-protocol>None</global-transactions-protocol>
        </data-source-params>
</data-source>
```

In the example, the data-source element's unique identifier is orads.

F.10 data-source-params

Use this element to specify data source-related data-source parameters.

F.10.1 Child Elements

The data-source-params server configuration element supports the child elements that Table F–5 lists.

Table F–5   Child Elements of: data-source-params

XML Tag	Type	Description
algorithm-type	See Description	The algorithm determines the connection request processing for the multi data source. Valid values: ■ Failover ■ Load-Balancing Default: Failover.
stream-chunk-size	int	Specifies the data chunk size for steaming data types between 1 and 65536. Default: 256.
row-prefetch	boolean	Specifies whether or not multiple rows to be prefetched (that is, sent from the server to the client) in one server access. Default: false.
data-source-list	string	The list of data sources to which the multi data source will route connection requests. The order of data sources in the list determines the failover order.
failover-request-if-busy	boolean	For multi data sources with the Failover algorithm, enables the multi data source to failover connection requests to the next data source if all connections in the current data source are in use.. Default: false.
row-prefetch-size	int	If row prefetching is enabled, specifies the number of result set rows to prefetch for a client between 2 and 65536. Default: 48.

Table F–5 (Cont.) Child Elements of: data-source-params

XML Tag	Type	Description
`jndi-names`	See Description	The JNDI path to where this Data Source is bound. By default, the JNDI name is the name of the data source. This element contains the following child elements: ■ `element`: contains the string name of a valid data-source element. For more information, see data-source. ■ `config-data-source-DataSourceParams-JNDINames`.
`scope`	`boolean`	Specifies the scoping of the data source. Note that `Global` is the only scoped supported by MSA. Default: `Global`.
`connection-pool-failover-callback-handler`	`string`	The name of the application class to handle the callback sent when a multi data source is ready to failover or fail back connection requests to another data source within the multi data source. The name must be the absolute name of an application class that implements the `weblogic.jdbc.extensions.ConnectionPoolFailoverCallback` interface.
`global-transactions-protocol`	`int`	Determines the transaction protocol (global transaction processing behavior) for the data source. Valid values: ■ `TwoPhaseCommit` - Standard XA transaction processing. Requires an XA driver ■ `LoggingLastResource` - A performance enhancement for one non-XA resource ■ `EmulateTwoPhaseCommit` - Enables one non-XA resource to participate in a global transaction, but has some risk to data ■ `OnePhaseCommit` - One-phase XA transaction processing using a non-XA driver. This is the default setting ■ `None` - Support for local transactions only Default: `OnePhaseCommit`.

F.10.2 Attributes

The `data-source-params` server configuration element has no attributes.

F.10.3 Example

The following example shows how to use the `data-source-params` element in the Oracle CEP server configuration file:

```
<data-source>
    <name>orads</name>
    <xa-params>
        <keep-xa-conn-till-tx-complete>true</keep-xa-conn-till-tx-complete>
    </xa-params>
    <driver-params>
        <url>jdbc:oracle:thin:@localhost:1521:ce102</url>
        <driver-name>oracle.jdbc.OracleDriver</driver-name>
        <properties>
            <element>
                <name>user</name>
                <value>wlevs</value>
            </element>
            <element>
                <name>password</name>
                <value>wlevs</value>
            </element>
        </properties>
    </driver-params>
    <connection-pool-params>
```

```
                    <initial-capacity>5</initial-capacity>
                    <max-capacity>10</max-capacity>
                    <test-table-name>SQL SELECT 1 FROM DUAL</test-table-name>
                    <test-frequency-seconds>5</test-frequency-seconds>
                </connection-pool-params>
                <data-source-params>
                    <jndi-names>
                        <element>orads</element>
                    </jndi-names>
                    <global-transactions-protocol>None</global-transactions-protocol>
                </data-source-params>
            </data-source>
```

F.11 driver-params

Use this element to specify JDBC driver-related `data-source` parameters.

F.11.1 Child Elements

The `driver-params` server configuration element supports the child elements that Table F–6 lists.

Table F–6   Child Elements of: driver-params

XML Tag	Type	Description
use-xa-data-source-interface	boolean	Specifies that the server should use the XA interface of the JDBC driver. If the JDBC driver class used to create database connections implements both XA and non-XA versions of a JDBC driver, you can set this attribute to indicate that the server should treat the JDBC driver as an XA driver or as a non-XA driver.. Default: `true`.
password	string	The password attribute passed to the JDBC driver when creating physical database connections..
driver-name	string	The full package name of JDBC driver class used to create the physical database connections in the connection pool in the data source..
url	string	The URL of the database to connect to. The format of the URL varies by JDBC driver. The URL is passed to the JDBC driver to create the physical database connections..
properties	string	Specifies the list of properties passed to the JDBC driver when creating physical database connections. This element contains one or more `element` child elements that contain child elements: ■ `name`: the prooperty name. ■ `value`: the property value.

F.11.2 Attributes

The `driver-params` server configuration element has no attributes.

F.11.3 Example

The following example shows how to use the `driver-params` element in the Oracle CEP server configuration file:

```
<data-source>
    <name>orads</name>
    <xa-params>
        <keep-xa-conn-till-tx-complete>true</keep-xa-conn-till-tx-complete>
    </xa-params>
    <driver-params>
        <url>jdbc:oracle:thin:@localhost:1521:ce102</url>
```

```
                    <driver-name>oracle.jdbc.OracleDriver</driver-name>
                    <properties>
                        <element>
                            <name>user</name>
                            <value>wlevs</value>
                        </element>
                        <element>
                            <name>password</name>
                            <value>wlevs</value>
                        </element>
                    </properties>
                </driver-params>
                <connection-pool-params>
                    <initial-capacity>5</initial-capacity>
                    <max-capacity>10</max-capacity>
                    <test-table-name>SQL SELECT 1 FROM DUAL</test-table-name>
                    <test-frequency-seconds>5</test-frequency-seconds>
                </connection-pool-params>
                <data-source-params>
                    <jndi-names>
                        <element>orads</element>
                    </jndi-names>
                    <global-transactions-protocol>None</global-transactions-protocol>
                </data-source-params>
            </data-source>
```

F.12 domain

Use this element to configure a domain name in the Oracle CEP server.

F.12.1 Child Elements

The domain server configuration element supports the following child elements:

- name

F.12.2 Attributes

The domain server configuration element has no attributes.

F.12.3 Example

The following example shows how to use the domain element in the Oracle CEP server configuration file:

```
<domain>
    <name>WLEventServerDomain</name>
</domain>
```

In the example, the domain's unique identifier is WLEventServerDomain.

F.13 debug

Use this element to configure one or more debug properties for the Oracle CEP server.

F.13.1 Child Elements

The debug server configuration element supports the child elements that Table F–7 lists.

Table F–7   Child Elements of: debug

XML Tag	Type	Description
name	string	The name of this debug configuration. For more information, see name.
debug-properties	string	One or more child elements formed by taking a debug flag name (without its package name) and specifying a value of true. For more information including a full list of all debug flags, see "How to Configure Oracle CEP Debugging Options Using a Configuration File" in the *Oracle CEP Administrator's Guide*

F.13.2 Attributes

The debug server configuration element has no attributes.

F.13.3 Example

The following example shows how to use the debug element to turn on Simple Declarative Services (SDS) debugging using debug flag com.bea.core.debug.DebugSDS in the Oracle CEP server configuration file.

```
<debug>
    <name>myDebug</name>
    <debug-properties>
        <DebugSDS>true</DebugSDS>
...
    </debug-properties>
</debug>
```

F.14 event-store

Use this element to configure an event store for the Oracle CEP server.

F.14.1 Child Elements

The event-store server configuration element supports the child elements that Table F–8 lists.

Table F–8   Child Elements of: event-store

XML Tag	Type	Description
name	string	The name of this debug configuration. For more information, see name.
provider-order	string	Specifies the name of one or more provider child elements in the order in which the Oracle CEP server should access them. For more information, see: ■ rdbms-event-store-provider ■ user-event-store-provider

F.14.2 Attributes

The event-store server configuration element has no attributes.

F.14.3 Example

The following example shows how to use the `event-store` element in the Oracle CEP server configuration file:

```
<config>
    <event-store>
        <name>myEventStore</name>
        <provider-order>
            <provider>provider1</provider>
            <provider>provider2</provider>
        </provider-order>
    </event-store>
</config>
```

In the example, the adapter's unique identifier is `myEventStore`.

F.15 exported-jndi-context

This configuration type is used to export a remote JNDI service that may be accessed via clients using RMI. It registers the JNDI context with the RMI service, so that it may be accessed remotely by clients that pass a provider URL parameter when they create their `InitialContext` object. This service requires that a `jndi-context` configuration object also be specified. If it is not, then this service will not be able to start.

F.15.1 Child Elements

The `exported-jndi-context` server configuration element supports the child elements that Table F–9 lists.

Table F–9   Child Elements of: exported-jndi-context

XML Tag	Type	Description
name	string	The name of this `debug` configuration. For more information, see `name`.
rmi-service-name	string	The name of the RMI service that should be used to serve this JNDI context over the network. It must match an existing RMI object in the configuration. For more information, see `rmi`.

F.15.2 Attributes

The `exported-jndi-context` server configuration element has no attributes.

F.15.3 Example

The following example shows how to use the `exported-jndi-context` element in the Oracle CEP server configuration file:

```
<rmi>
    <name>myRMI</name>
    <http-service-name>TestJetty</http-service-name>
</rmi>

<exported-jndi-context>
    <name>RemoteJNDI</name>
    <rmi-service-name>myRMI</rmi-service-name>
</exported-jndi-context>
```

In the example, the adapter's unique identifier is `RemoteJNDI`.

F.16 http-pubsub

Use this element to configure an HTTP publish-subscribe service.

F.16.1 Child Elements

The `http-pubsub` server configuration element supports the following child elements:

- `name`
- `path`
- `pubsub-bean`

F.16.2 Attributes

The `http-pubsub` server configuration element has no attributes.

F.16.3 Example

The following example shows how to use the `http-pubsub` element in the Oracle CEP server configuration file:

```
<http-pubsub>
    <name>myPubsub</name>
    <path>/pubsub</path>
    <pub-sub-bean>
        <server-config>
            <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>
```

In the example, the `http-pubsub` element's unique identifier is `myPubsub`.

F.17 jetty

Use this element to configure an instance of the Jetty HTTP server.

F.17.1 Child Elements

The `jetty` server configuration element supports the child elements that Table F–10 lists.

Table F–10   Child Elements of: jetty

XML Tag	Type	Description
name	string	The name of this `jetty` element. For more information, see name.
network-io-name	string	The name of the Network I/O service that should be used. This also defines which port the server listens on. This parameter must refer to the name of a valid "netio" configuration object.
work-manager-name	string	The name of the Work Manager that should be used for thread pooling. If this parameter is not specified, then Jetty will use a default work manager. For more information, see work-manager.
scratch-directory	string	The name of a directory where temporary files required for Web applications, JSPs, and other types of Web artifacts are kept. This parameter is overridden by the `scratch-directory` parameter on the `jetty-web-app` element. If this directory does not exist, it will be created.
debug-enabled	boolean	Enable debugging in the Jetty code. Specified debug messages are logged the same way as all other `Debug` level messages in the log service.
listen-port	int	The name of the network port that should be set. This parameter may not be set if the `network-io-name` parameter is not specified. When this parameter is used instead of `network-io-name`, a simplified socket I/O subsystem is used that does not require the `netio` module.
secure-network-io-name	string	The name of the Network I/O service that should be used for secure communications. The specified service must be configured to support SSL encryption. This parameter must refer to the name of a valid `netio` configuration object.

F.17.2 Attributes

The `jetty` server configuration element has no attributes.

F.17.3 Example

The following example shows how to use the `jetty` element in the Oracle CEP server configuration file:

```
<jetty>
    <name>TestJetty</name>
    <work-manager-name>WM</work-manager-name>
    <network-io-name>Netio</network-io-name>
    <secure-network-io-name>SecureNetio</secure-network-io-name>
    <debug-enabled>false</debug-enabled>
    <scratch-directory>JettyWork</scratch-directory>
</jetty>
```

In the example, the `jetty` element's unique identifier is `TestJetty`.

F.18 jetty-web-app

Use this element to represent a Web application for use by Jetty. Each instance of this object represents a Web application which must be deployed using the Jetty service.

F.18.1 Child Elements

The `jetty-web-app` server configuration element supports the child elements that Table F–11 lists.

Table F–11   Child Elements of: jetty-web-app

XML Tag	Type	Description
name	string	The name of this `jetty-web-app` element. For more information, see `name`.
context-path	string	The context path where this web app will be deployed in the web server's name space. Default:`/`
scratch-directory	string	The location where Jetty should store temporary files for this web app. This parameter overrides the `scratch-directory` parameter on the `jetty` element. If this directory does not exist, it will be created.
path	string	A file name that points to the location of the web app on the server. It may be a directory or a WAR file.
jetty-name	string	The name of the Jetty service where this Web application should be deployed. This name must match the name of an existing `jetty` configuration object. For more information, see `jetty`.

F.18.2 Attributes

The `jetty-web-app` server configuration element has no attributes.

F.18.3 Example

The following example shows how to use the `jetty-web-app` element in the Oracle CEP server configuration file:

```
<jetty-web-app>
    <name>financial</name>
    <context-path>/financial</context-path>
    <path>../testws2/financialWS.war</path>
    <jetty-name>TestJetty</jetty-name>
</jetty-web-app>
```

In the example, the `jetty-web-app` element's unique identifier is `financial`.

F.19 jmx

Use this element to configure Java Management Extension (JMX) properties in the Oracle CEP server.

F.19.1 Child Elements

The `jmx` server configuration element supports the child elements that Table F–12 lists.

Table F–12   Child Elements of: jmx

XML Tag	Type	Description
name	string	The name of this `debug` configuration. For more information, see `name`.
rmi-service-name	string	The name of the RMI service that should be used to serve this JNDI context over the network. It must match an existing RMI object in the configuration. For more information, see `rmi`.
jndi-service-name	string	The name of the JNDI service to which the JMX server will bind its object.

F.19.2 Attributes

The `jmx` server configuration element has no attributes.

F.19.3 Example

The following example shows how to use the `jmx` element in the Oracle CEP server configuration file:

```
<jmx>
    <name>myJMX</name>
    <jndi-service-name>JNDI</jndi-service-name>
    <rmi-service-name>RMI</rmi-service-name>
</jmx>
```

In the example, the `jmx` element's unique identifier is `myJMX`.

F.20 jndi-context

This configuration object is used to configure the JNDI provider. When it is placed in the configuration, the MSA JNDI Context is initialized. One instance of this configuration type must be placed in the configuration if the JNDI service is to be used, either locally, or remotely through the exported-jndi-context configuration type..

F.20.1 Child Elements

The `jndi-context` server configuration element supports the child elements that Table F–13 lists.

Table F–13   Child Elements of: jndi-context

XML Tag	Type	Description
name	string	The name of this `debug` configuration. For more information, see `name`.
default-provider	string	This parameter defaults to `true`. If it is set to `false` then the provider will not be installed as the default. The provider will be set as the default as long as there is at least one `JNDIContextType` bean in the configuration with `DefaultProvider` set to `true`. If multiple `JNDIContextType` objects are placed in the configuration, the effect will be the same as if one was started.

F.20.2 Attributes

The `jndi-context` server configuration element has no attributes.

F.20.3 Example

The following example shows how to use the `jndi-context` element in the Oracle CEP server configuration file:

```
<jndi-context>
    <name>myJNDI</name>
    <default-provider>true</default-provider>
</jndi-context>
```

In the example, the adapter's unique identifier is `myJNDI`.

F.21 log-file

Use this element to configure logging to a file on the Oracle CEP server.

F.21.1 Child Elements

The `log-file` server configuration element supports the child elements that
Table F–14 lists.

Table F–14   Child Elements of: log-file

XML Tag	Type	Description
name	string	The name of this `work-manager` element. For more information, see name.
number-of-files-limited	boolean	Determines whether old rotated files need to be kept around forever. Default: `false`.
rotation-type	string	Determines how the log file rotation will be performed based on size, time or not at all. Valid values: ■　bySize ■　byTime ■　none Default: `bySize`.
rotation-time	string	The time in `k:mm` format when the first rotation happens where `k` is the hour specified in 24-hour notation and `mm` is the minutes. Default: 00:00.
rotated-file-count	int	If old rotated files are to be deleted, this parameter determines how many of the last files to always keep. Default: 7.
rotation-size	int	The size threshold at which the log file is rotated in KB. Default: 500.
rotation-time-span-factor	long	The factor that is applied to the timespan to arrive at the number of milliseconds that will be the frequency of time based log rotations. Default: (`long`)(3600 * 1000).
rotation-time-span	int	The interval for every time based log rotation. Default: 24.
base-log-file-name	string	Log file name. Default: `server.log`
rotate-log-on-startup-enabled	boolean	Specifies whether the log file will be rotated on startup. Default: `true`.

Table F–14   (Cont.) Child Elements of: log-file

XML Tag	Type	Description
log-file-severity	string	Specifies the threshold importance of the messages that are propagated to the handlers. The default is Info so that to see Debug and Trace messages you need to ensure that the severity is set to either Debug or Trace. Valid values: ■　Emergency ■　Alert ■　Critical ■　Error ■　Warning ■　Notice ■　Info ■　Debug ■　Trace Default: Notice.
log-file-rotation-dir	string	The directory where the old rotated files are stored. If not set, the old files are stored in the same directory as the base log file.

F.21.2 Attributes

The log-file server configuration element has no attributes.

F.21.3 Example

The following example shows how to use the log-file element in the Oracle CEP server configuration file:

```
<log-file>
    <name>logFile</name>
    <number-of-files-limited>true</number-of-files-limited>
    <rotated-file-count>4</rotated-file-count>
    <rotate-log-on-startup-enabled>true</rotate-log-on-startup-enabled>
</log-file>
```

In the example, the log-file element's unique identifier is logFile.

F.22 log-stdout

Use this element to configure logging to standard out (console) on the Oracle CEP server.

F.22.1 Child Elements

The log-stdout server configuration element supports the child elements that Table F–15 lists.

Table F–15   Child Elements of: log-stdout

XML Tag	Type	Description
name	string	The name of this work-manager element. For more information, see name.
stack-trace-depth	int	Determines the number of stack trace frames to display on standard out. All frames are displayed in the log file. A value of -1 means all frames are displayed. Default: -1.

Table F–15   (Cont.) Child Elements of: log-stdout

XML Tag	Type	Description
stack-trace-enabled	boolean	Specifies whether to dump stack traces to the console when included in a logged message. Default: true.
stdout-severity	string	Defines the threshold importance of the messages that are propagated to the handlers. The default is Info so that to see Debug and Trace messages you need to ensure that the severity is set to either Debug or Trace. Valid values: ■ Emergency ■ Alert ■ Critical ■ Error ■ Warning ■ Notice ■ Info ■ Debug ■ Trace Default: Notice.

F.22.2 Attributes

The log-stdout server configuration element has no attributes.

F.22.3 Example

The following example shows how to use the log-stdout element in the Oracle CEP server configuration file:

```
<log-stdout>
    <name>logStdout</name>
    <stdout-severity>Debug</stdout-severity>
</log-stdout>
```

In the example, the log-stdout element's unique identifier is logStdout.

F.23 logging-service

Use this element to configure a logging service on the Oracle CEP server.

F.23.1 Child Elements

The logging-service server configuration element supports the child elements that Table F–16 lists.

Table F–16   Child Elements of: logging-service

XML Tag	Type	Description
name	string	The name of this work-manager element. For more information, see name.
log-file-config	string	The configuration of the log file and its rotation policies.
stdout-config	string	The configuration of the stdout output.

Table F–16   (Cont.) Child Elements of: logging-service

XML Tag	Type	Description
logger-severity	string	Defines the threshold importance of the messages that are propagated to the handlers. The default is `Info` so that to see `Debug` and `Trace` messages you need to ensure that the severity is set to either `Debug` or `Trace`. Valid values: ■ Emergency ■ Alert ■ Critical ■ Error ■ Warning ■ Notice ■ Info ■ Debug ■ Trace Default: `Info`.
logger-severity-properties	See Description	The Severity values for different nodes in the Logger tree composed of one or more `entry` child elements each containing a `key` and `value` child element.

F.23.2 Attributes

The `logging-service` server configuration element has no attributes.

F.23.3 Example

The following example shows how to use the `logging-service` element in the Oracle CEP server configuration file:

```
<logging-service>
    <name>myLogService</name>
    <stdout-config>myStdoutConfig</stdout-config>
    <logger-severity>Notice</logger-severity>
    <logger-severity-properties>
      <entry>
        <key>FileAdapter</key>
        <value>Debug</value>
      </entry>
      <entry>
        <key>CQLProcessor</key>
        <value>Debug</value>
      </entry>
    </logger-severity-properties>
</logging-service>
```

In the example, the `logging-service` element's unique identifier is `myLogService`.

F.24 message-filters

Use this element to configure one or more message filters for a `pubsub-bean` element.

F.24.1 Child Elements

The `message-filters` server configuration element contains one or more `element` child elements that each contain a `message-filter-name` and `message-filter-class` child element.

F.24.2 Attributes

The `message-filters` server configuration element has no attributes.

F.24.3 Example

The following example shows how to use the `message-filters` element in the Oracle CEP server configuration file:

```
<http-pubsub>
    <name>pubsub</name>
    <path>/pubsub</path>
    <pub-sub-bean>
...
      <message-fitlers>
        <element>
          <message-filter-name>Foo</message-filter-name>
          <message-filter-class>Foo</message-filter-class>
        </element>
        <element>
          <message-filter-name>Foo</message-filter-name>
          <message-filter-class>Foo</message-filter-class>
        </element>
      </message-filters>
...
    </pub-sub-bean>
</http-pubsub>
```

F.25 name

Use this element to declare a unique identifier for an Oracle CEP server configuration element.

F.25.1 Child Elements

The `name` server configuration element has no child elements.

F.25.2 Attributes

The `name` server configuration element has no attributes.

F.25.3 Example

The following example shows how to use the `name` element in the Oracle CEP server configuration file:

```
<http-pubsub>
    <name>pubsub</name>
    <path>/pubsub</path>
...
</http-pubsub>
```

F.26 netio

Use this element to represent a Netio service, which may be used by other services to act as the server for network input/output.

F.26.1 Child Elements

The `netio` server configuration element supports the child elements that Table F–17 lists.

Table F–17   Child Elements of: netio

XML Tag	Type	Description
name	string	The name of this `netio` element. For more information, see name.
ssl-config-bean-name	string	The name of the SSL configuration object to use. If not null, then this client will create secure sockets using the specified SSL configuration. If not set, then no SSL will be supported.
provider-type	string	Specify which provider to use for the underlying socket implementation. See the MSA documentation for a list of the valid provider types.
io-threads	int	A hint to the provider as to the number of threads to use for processing sockets. A value of zero will result in the provider choosing based on its own default. Default: 0.
port	int	The port to listen on. The server will immediately start to listen for incoming connections on this port.
listen-address	string	The address on which this instance of Netio should listen for incoming connections. It may be set to a numeric IP address in the `a.b.c.d` format, or to a host name. If not set, then Netio will listen on all network interfaces. Note that the value of this parameter cannot be validated until the server actually starts.

F.26.2 Attributes

The `netio` server configuration element has no attributes.

F.26.3 Example

The following example shows how to use the `netio` element in the Oracle CEP server configuration file:

```
<netio>
    <name>myNetio</name>
    <port>12345</port>
</netio>
```

In the example, the `netio` element's unique identifier is `myNetio`.

F.27 netio-client

Use this element to register a NetIO service that may be used to perform non-blocking network I/O, but which will not act as a server and listen for incoming connections.

F.27.1 Child Elements

The `netio-client` server configuration element supports the child elements that Table F–18 lists.

Table F–18   Child Elements of: netio-client

XML Tag	Type	Description
name	string	The name of this `netio` element. For more information, see name.

Table F–18   (Cont.) Child Elements of: netio-client

XML Tag	Type	Description
ssl-config-bean-name	string	The name of the SSL configuration object to use. If not null, then this client will create secure sockets using the specified SSL configuration. If not set, then no SSL will be supported.
provider-type	string	Specify which provider to use for the underlying socket implementation. See the MSA documentation for a list of the valid provider types.

F.27.2 Attributes

The `netio-client` server configuration element has no attributes.

F.27.3 Example

The following example shows how to use the `netio-client` element in the Oracle CEP server configuration file:

```
<netio-client>
    <name>netiossl</name>
    <ssl-config-bean-name>sslConfig</ssl-config-bean-name>
    <provider-type>NIO</provider-type>
</netio-client>
```

In the example, the `netio-client` element's unique identifier is `netiossl`.

F.28 path

Use this element to configure the path for an `http-pubsub` element.

F.28.1 Child Elements

The `path` element has no child elements.

F.28.2 Attributes

The `path` element has no attributes.

F.28.3 Example

The following example shows how to use the `path` element in the Oracle CEP server configuration file:

```
<http-pubsub>
    <name>myPubsub</name>
    <path>/pubsub</path>
    <pub-sub-bean>
        <server-config>
            <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>
```

F.29 pubsub-bean

Use this element to configure a publish-subscribe bean for an `http-pubsub` element.

F.29.1 Child Elements

The `pubsub-bean` server configuration element supports the following child elements:

- `name`

- `server-config`

- `message-filters`

- `channels`

- `channel-constraints`

- `services`

F.29.2 Attributes

The `pubsub-bean` server configuration element has no attributes.

F.29.3 Example

The following example shows how to use the `pubsub-bean` element in the Oracle CEP server configuration file:

```
<http-pubsub>
    <name>myPubsub</name>
    <path>/pubsub</path>
    <pub-sub-bean>
        <server-config>
            <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>
```

F.30 rdbms-event-store-provider

Use this element to configure an event store provider that uses a relational database management system in the Oracle CEP server.

F.30.1 Child Elements

The `rdbms-event-store-provider` server configuration element supports the child elements that Table F–19 lists.

Table F–19   Child Elements of: rdbms-event-store-provider

XML Tag	Type	Description
name	string	The name of this debug configuration. For more information, see name.
init-timeout	int	The maximum time (in milliseconds) that the Oracle CEP server will wait for this provider to initialize. Default: 10000 ms.
data-source-name	string	The name of a data source element. For more information, see data-source.
user-policy-attributes	See Description	One or more entry child elements that each contain a key and value child element that you use to specify additional data source properties.

F.30.2 Attributes

The `rdbms-event-store-provider` server configuration element has no attributes.

F.30.3 Example

The following example shows how to use the `rdbms-event-store-provider` element in the Oracle CEP server configuration file:

```
<rdbms-event-store-provider>
    <name>test-rdbms-provider</name>
    <init-timeout>10000</init-timeout>
    <data-source-name>derby1</data-source-name>
    <user-policy-attributes>
        <entry>
            <key>key1</key>
            <value>value1</value>
        </entry>
            <key>key1</key>
            <value>value1</value>
        <entry>
        </entry>
    </user-policy-attributes>
</rdbms-event-store-provider>
```

In the example, the `rdbms-event-store-provider` element's unique identifier is `test-rdbms-provider`.

F.31 rmi

Use this element to configure an RMI service, which allows server- side objects to be exported to remote clients.

F.31.1 Child Elements

The `rmi` server configuration element supports the child elements that Table F–20 lists.

Table F–20   Child Elements of: rmi

XML Tag	Type	Description
name	string	The name of this `rmi` element. For more information, see name.
heartbeat-period	int	The number of failed heartbeat attempts before triggering disconnect notifications to all registered listeners. Default-Value: 4.
http-service-name	string	The name of the HTTP service that this service should use to register remote objects. The service may be provided by a Jetty or Tomcat instance of the same name.
heartbeat-interval	int	The time in milliseconds between heartbeats. Once the number of unsuccessful heartbeat attempts has reached the value specified by the `HeartbeatPeriod` attribute, all registered `DisconnectListener` instances will be notified. Default-Value: 5000.

F.31.2 Attributes

The `rmi` server configuration element has no attributes.

F.31.3 Example

The following example shows how to use the `rmi` element in the Oracle CEP server configuration file:

```
<rmi>
    <name>myRMI</name>
    <http-service-name>TestJetty</http-service-name>
</rmi>
```

In the example, the `rmi` element's unique identifier is `myRMI`.

F.32 scheduler

Use this element to configure `cql` scheduler options in the Oracle CEP server.

F.32.1 Child Elements

The `scheduler` server configuration element supports the child elements that Table F–21 lists.

Table F–21   Child Elements of: scheduler

XML Tag	Type	Description
class-name	string	Specify the value for one of the sched_name scheduling option as the fully qualified package name of Java class that implements the Oracle CEP Service Engine scheduling algorithm. This class determines in what order the Oracle CEP Service Engine scheduler executes Oracle CQL queries. Valid values: ■ oracle.cep.execution.scheduler.RoundRobinScheduler : This algorithm assigns time slices to each Oracle CQL query in equal portion and in order, handling all processes without priority. This option is appropriate if the number of Oracle CQL queries is not prone to large variations. ■ oracle.cep.execution.scheduler.FIFOScheduler: This algorithm assigns time slices to each Oracle CQL query in the order that they were created. This algorithm is appropriate if the number of Oracle CQL queries is prone to large variations. Default: oracle.cep.execution.scheduler.RoundRobinScheduler
runtime	long	Total number of seconds that the Oracle CEP Service Engine scheduler will run. Default: 1000000 ms.
time-slice	int	The frequency at which the Oracle CEP Service Engine scheduler executes Oracle CQL queries. Default: 1000 ms
schedule-on-new-thread	boolean	Whether or not the Oracle CEP Service Engine scheduler will use a separate thread. Options are: ■ true: the scheduler runs in a separate thread. ■ false: the scheduler runs in the same thread as the Oracle CEP Service Engine (Default).

F.32.2 Attributes

The scheduler server configuration element has no attributes.

F.32.3 Example

The following example shows how to use the scheduler element in the Oracle CEP server configuration file:

```
<cql>
    <name>myCQL</name>
    <scheduler>
        <class-name>oracle.cep.execution.scheduler.FIFOScheduler</class-name>
    </scheduler>
</cql>
```

F.33 server-config

Use this element to configure the server-specific properties of a pubsub-bean element.

F.33.1 Child Elements

The server-config server configuration element supports the child elements that Table F–22 lists.

Table F–22   Child Elements of: server-config

XML Tag	Type	Description
name	string	The name of this `server-config` element. For more information, see name.
supported-transport	See Description	This element contains one or more `types` child elements, one for each supported transport. Each `types` child element contains an `element` child element with the transport name as a `string` value. Valid values: ■ `long-polling`: Using this transport, the client requests information from Oracle CEP server and if Oracle CEP server does not have information available, it does not reply until it has. When the Oracle CEP server replies, the client typically sends another request immediately. ■ `callback-polling`: Use this transport for HTTP publish-subscribe applications using a cross domain configuration in which the browser downloads the page from one Web server (including the Javascript code) and connects to another server as an HTTP publish-subscribe client. This is required by the Bayeaux protocol. For more information on the Bayeaux protocol, see `http://svn.xantus.org/shortbus/trunk/bayeux/bayeux.html`. For more information, see "How the HTTP Pub-Sub Server Works" in the *Oracle CEP Administrator's Guide*.
client-timeout-secs	int	Specifies the number of seconds after which the HTTP pub-sub server disconnects a client if the client does has not sent back a connect/reconnect message. Default: 60.
persistent-client-timeout-secs	int	Specifies the number of seconds after which persistent clients are disconnected and deleted by the pub-sub server, if during that time the persistent client does not send a connect or re-connect message. This value must be larger than client-timeout-secs. If the persistent client reconnects before the persistent timeout is reached, the client receives all messages that have been published to the persistent channel during that time; if the client reconnects after the timeout, then it does not get the messages. Default: 600 seconds.
interval-millisecs	int	Specifies how long (in milliseconds) the client can delay subsequent requests to the `/meta/connect` channel. Default: 500 ms.
work-manager	string	Specifies the name of the work manager that delivers messages to clients. The value of this element corresponds to the value of the `name` child element of the `work-manager` you want to assign. For more information, see work-manager.
publish-without-connect-allowed	boolean	Specifies whether clients can publish messages without having explicitly connected to the HTTP pub-sub server. Valid values: ■ `true` ■ `false`

F.33.2 Attributes

The `server-config` server configuration element has no attributes.

F.33.3 Example

The following example shows how to use the `server-config` element in the Oracle CEP server configuration file:

```
<http-pubsub>
```

```
                    <name>pubsub</name>
                    <path>/pubsub</path>
                    <pub-sub-bean>
                      <server-config>
                        <name>/pubsub</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>
```

F.34 services

Use this element to configure the service properties of a pubsub-bean element.

F.34.1 Child Elements

The services server configuration element contains one or more element child elements that each support the child elements that Table F–23 lists.

Table F–23   Child Elements of: services

XML Tag	Type	Description
service-channel	string	Specifies a service channel, for example: /service/echo.
service-class	string	Specifies the class to service this service, for example: EchoService.
service-method	string	Define a service method in the service class. The service method must have only one payload parameter of type Object. For example: Object echo(Object payload).

F.34.2 Attributes

The services server configuration element has no attributes.

F.34.3 Example

The following example shows how to use the services element in the Oracle CEP server configuration file:

```
<http-pubsub>
    <name>pubsub</name>
    <path>/pubsub</path>
    <pub-sub-bean>
      <server-config>
```

```
            <name>/pubsub</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>
      <services>
          <element>
              <service-channel>Foo</service-channel>
              <service-class>Foo</service-class>
              <service-method>Foo</service-method>
          </element>
      </services>
      </pub-sub-bean>
</http-pubsub>
```

F.35 show-detail-error-message

Use this element to configure whether or not the Oracle CEP server uses secure
connections.

F.35.1 Child Elements

The `show-detail-error-message` server configuration element supports the child
elements that Table F–24 lists.

Table F–24   Child Elements of: show-detail-error-message

XML Tag	Type	Description
name	string	The name of this `show-detail-error-message` element. For more information, see name.
value	boolean	Whether or not to show detailed error messages. Valid values: ■　　`true`: the Oracle CEP server shows detailed error messages. ■　　`false`: the Oracle CEP server shows abbreviated error messages (default).

F.35.2 Attributes

The `show-detail-error-message` server configuration element has no attributes.

F.35.3 Example

The following example shows how to use the `show-detail-error-message`
element in the Oracle CEP server configuration file:

```
<show-detail-error-message>
```

```
        <name>myShowDetail</name>
        <value>true</value>
</show-detail-error-message>
```

In the example, the `show-detail-error-message` element's unique identifier is `myShowDetail`.

F.36 ssl

Use this element to configure Secure Sockets Layer-specific properties on the Oracle CEP server.

F.36.1 Child Elements

The `ssl` server configuration element supports the child elements that Table F–25 lists.

Table F–25   Child Elements of: ssl

XML Tag	Type	Description
name	string	The name of this cluster. For more information, see name.
key-store	string	Specifies the file path to the key store such as `./ssl/evsidentity.jks`.
key-store-pass	See Description	This element contains a `password` child element with a `string` value that specifies the password used to access the key store.
key-store-alias	string	Specifies the alias for the key store.
key-manager-algorithm	string	Specifies the key manager algorithm such as `SunX509`.
ssl-protocol	string	Specifies the SSL protocol such as `TLS`.
trust-store	string	Specifies the file path to the trust store such as `./ssl/evstrust.jks`.
trust-store-pass	See Description	This element contains a `password` child element with a `string` value that specifies the password used to access the trust store.
trust-store-alias	string	Specifies the alias for the trust store.
trust-store-type	string	Specifies the trust store type such as `JKS`.
trust-manager-algorithm	string	Specifies the trust manager algorithm such as `SunX509`.
enforce-fips	boolean	Specifies whether or not Oracle CEP server uses a Federal Information Processing Standards (FIPS)-certified pseudo-random number generator. For more information, see "FIPS" in the *Oracle CEP Administrator's Guide*.
need-client-auth	boolean	Specifies whether or not client certificate authentication is required.
ciphers	See Description	This element contains one or more `cipher` child elements, each with a `string` value that specifies the ciphers that are required.
secure-random-algorithm	string	When `enforce-fips` is set to `true`, specify the secure random algorithm to use. Valid values: ■ FIPS186PRNG
secure-random-provider	string	When `enforce-fips` is set to `true`, specify the secure random provider to use. Valid values: ■ JsafeJCE

F.36.2 Attributes

The `ssl` server configuration element has no attributes.

F.36.3 Example

The following example shows how to use the `ssl` element in the Oracle CEP server configuration file:

```
<ssl>
    <name>sslConfig</name>
    <key-store>./ssl/evsidentity.jks</key-store>
    <key-store-pass>
        <password>{Salted-3DES}s4YUEvH4Wl2DAjb45iJnrw==</password>
    </key-store-pass>
    <key-store-alias>evsidentity</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>
```

In the example, the `ssl` element's unique identifier is `sslConfig`.

F.37 timeout-seconds

Use this element to configure `weblogic-jta-gateway` default transaction timeout in seconds in the Oracle CEP server.

Default: 60.

F.37.1 Child Elements

The `timeout-seconds` server configuration element has no attributes.

F.37.2 Attributes

The `timeout-seconds` server configuration element has no attributes.

F.37.3 Example

The following example shows how to use the `timeout-seconds` element in the Oracle CEP server configuration file:

```
<weblogic-jta-gateway>
    <name>myJTAGateway</name>
    <timeout-seconds>90</timeout-seconds>
    <weblogic-instances>
        <weblogic-instance>
            <domain-name>ocep_domain</domain-name>
            <server-name>fxserver</server-name>
            <protocol>t3</protocol>
            <host-address>ariel</host-address>
            <port>9002</port>
        </weblogic-instance>
    </weblogic-instances>
</weblogic-jta-gateway>
```

F.38 transaction-manager

Use this element to configure transaction manager properties in the Oracle CEP server.

F.38.1 Child Elements

The `transaction-manager` server configuration element supports the child elements that Table F–26 lists.

Table F–26   Child Elements of: transaction-manager

XML Tag	Type	Description
name	string	The name of this `transaction-manager` element. For more information, see name.
max-resource-requests-on-server	int	Maximum number of concurrent requests to resources allowed for each server. Default: 50.
max-resource-unavailable-millis	long	Maximum duration in milliseconds that a resource is declared dead. After the duration, the resource will be declared available again, even if the resource provider does not explicitly re-register the resource. Default: 1800000.
security-interop-mode	string	Specifies the security mode of the communication channel used for XA calls between servers that participate in a global transaction. All server instances in a domain must have the same security mode setting. Valid values: `default`: The transaction coordinator makes calls using the kernel identity over an admin channel if it is enabled, and anonymous otherwise. Man-in-the-middle attacks are possible if the admin channel is not enabled. `Performance`: The transaction coordinator makes calls using anonymous at all times. This implies a security risk since a malicious third party could then try to affect the outcome of transactions using a man-in-the-middle attack. `Compatibility`: The transaction coordinator makes calls as the kernel identity over an insecure channel. This is a high security risk because a successful man-in-the-middle attack would allow the attacker to gain administrative control over both domains. This setting should only be used when strong network security is in place. Default: `default`.
parallel-xa-enabled	boolean	Execute XA calls in parallel if there are available threads. Default: `true`.
tlog-location	string	The location of the file store that contains the transaction log. This attribute can be either an absolute or relative path in the filesystem.
max-xa-call-millis	long	Maximum allowed duration of XA calls to resources. If a particular XA call to a resource exceeds the limit, the resource is declared unavailable. Default: 120000.
timeout-seconds	int	The default transaction timeout in seconds. Default: 30.
checkpoint-interval-seconds	int	The interval at which the transaction manager performs transaction log checkpoint operations. Default: 300.

Table F–26   (Cont.) Child Elements of: transaction-manager

XML Tag	Type	Description
forget-heuristics	boolean	Specifies whether the transaction manager will automatically perform an XAResource forget operation for heuristic transaction completions. When enabled, the transaction manager automatically performs an XA Resource `forget()` operation for all resources as soon as the transaction learns of a heuristic outcome. Disable this feature only if you know what to do with the resource when it reports a heuristic decision. Default: `true`.
before-completion-iteration-limit	int	The maximum number of cycles that the transaction manager will perform the before completion synchronization callback processing. Default: 10.
abandon-timeout-seconds	int	The transaction abandon timeout seconds for transactions in the second phase of the two-phase commit (prepared and later). During the second phase of the two-phase commit process, the transaction manager will continue to try to complete the transaction until all resource managers indicate that the transaction is completed. Using this timeout, you can set the maximum time that a transaction manager will persist in attempting to complete a transaction during the second phase of the transaction. After the abandon transaction timer expires, no further attempt is made to resolve the transaction. If the transaction is in a prepared state before being abandoned, the transaction manager will roll back the transaction to release any locks held on behalf of the abandoned transaction. Default: 86400.
serialize-enlistments-gc-interval-millis	long	The interval at which internal objects used to serialize resource enlistment are cleaned up. Default: 30000.
unregister-resource-grace-period	int	The grace period (number of seconds) that the transaction manager waits for transactions involving the resource to complete before unregistering a resource. The grace period can help minimize the risk of abandoned transactions because of an unregistered resource, such as a JDBC data source module packaged with an application. During the specified grace period, the unregisterResource call will block until the call can return, and no new transactions are started for the associated resource. If the number of outstanding transactions for the resource goes to 0, the unregisterResource call returns immediately. At the end of the grace period, if there are still outstanding transactions associated with the resource, the unregisterResource call returns and a log message is written on the server on which the resource was previously registered. Default: 30.
rmi-service-name	string	The name of the RMI service that is used for distributed transaction coordination. For more information, see rmi.
max-unique-name-statistics	int	The maximum number of unique transaction names for which statistics will be maintained. Default: 1000.
purge-resource-from-checkpoint-interval-seconds	int	The interval that a particular resource must be accessed within for it to be included in the checkpoint record. Default: 86400.
max-transactions	int	The maximum number of simultaneous in-progress transactions allowed on this server. Default: 10000.
migration-checkpoint-interval-seconds	int	The interval that the checkpoint is done for the migrated transaction logs (TLOGs). Default: 60.

Table F–26 (Cont.) Child Elements of: transaction-manager

XML Tag	Type	Description
recovery-thresho ld-millis	long	The interval that recovery is attempted until the resource becomes available. Default: 300000.
max-transactions -health-interval -millis	long	The interval for which the transaction map must be full for the JTA subsystem to declare its health as CRITICAL. Default: 60000.
parallel-xa-disp atch-policy	string	The dispatch policy to use when performing XA operations in parallel. By default the policy of the thread coordinating the transaction is used.

F.38.2 Attributes

The transaction-manager server configuration element has no attributes.

F.38.3 Example

The following example shows how to use the transaction-manager element in the Oracle CEP server configuration file:

```
<transaction-manager>
    <name>My_tm</name>
    <timeout-seconds>30</timeout-seconds>
    <abandon-timeout-seconds>86400</abandon-timeout-seconds>
    <forget-heuristics>true</forget-heuristics>
    <before-completion-iteration-limit>12</before-completion-iteration-limit>
    <max-transactions>10100</max-transactions>
    <max-unique-name-statistics>500</max-unique-name-statistics>
    <max-resource-requests-on-server>50</max-resource-requests-on-server>
    <max-resource-unavailable-millis>1800000</max-resource-unavailable-millis>
    <recovery-threshold-millis>300000</recovery-threshold-millis>
    <max-transactions-health-interval-millis>
        60000
    </max-transactions-health-interval-millis>
    <purge-resource-from-checkpoint-interval-seconds>
        86400
    </purge-resource-from-checkpoint-interval-seconds>
    <checkpoint-interval-seconds>300</checkpoint-interval-seconds>
    <parallel-xa-enabled>true</parallel-xa-enabled>
    <unregister-resource-grace-period>30</unregister-resource-grace-period>
    <security-interop-mode>default</security-interop-mode>
    <rmi-service-name>RMI_ce1</rmi-service-name>
  </transaction-manager>
```

In the example, the transaction-manager element's unique identifier is My_tm.

F.39 use-secure-connections

Use this element to configure whether or not the Oracle CEP server uses secure connections.

For more information, see "How to Configure SSL in a Multi-Server Domain for Oracle CEP Visualizer" in the *Oracle CEP Administrator's Guide*.

F.39.1 Child Elements

The `use-secure-connections` server configuration element supports the child elements that Table F–27 lists.

Table F–27   Child Elements of: use-secure-connections

XML Tag	Type	Description
name	string	The name of this `use-secure-connections` element. For more information, see name.
value	boolean	Whether or not to use secure connections. Valid values:
		■ `true`: the Oracle CEP server uses only secure connections.
		■ `false`: the Oracle CEP server accepts connections that are not secure.

F.39.2 Attributes

The `use-secure-connections` server configuration element has no attributes.

F.39.3 Example

The following example shows how to use the `use-secure-connections` element in the Oracle CEP server configuration file:

```
<use-secure-connections>
    <name>myUseSecConn</name>
    <value>true</value>
</use-secure-connections>
```

In the example, the `use-secure-connections` element's unique identifier is `myUseSecConn`.

F.40 user-event-store-provider

Use this element to configure an event store provider in the Oracle CEP server that is implemented by a Java class that you provide.

F.40.1 Child Elements

The `user-event-store-provider` server configuration element supports the child elements that Table F–28 lists.

Table F–28   Child Elements of: user-event-store-provider

XML Tag	Type	Description
name	string	The ID of the bean element that defines the Java class that implements this `user-event-store-provider`. For more information, see name.
init-timeout	int	The maximum time (in milliseconds) that the Oracle CEP server will wait for this provider to initialize.
		Default: 10000 ms.

F.40.2 Attributes

The `user-event-store-provider` server configuration element has no attributes.

F.40.3 Example

The following example shows how to use the `user-event-store-provider` element in the Oracle CEP server configuration file:

```
<user-event-store-provider>
    <name>test-user-provider</name>
    <init-timeout>10000</init-timeout>
</user-event-store-provider>
```

In the example, the user-event-store-provider element's unique identifier is `test-user-provider`.

F.41 weblogic-instances

Use this element to configure Oracle CEP server instances for a `weblogic-jta-gateway` element.

F.41.1 Child Elements

The `weblogic-instances` server configuration element supports zero or more `weblogic-instance` child elements that each contain the child elements that Table F–29 lists.

Table F–29   Child Elements of: weblogic-instances

XML Tag	Type	Description
domain-name	string	Specifies the name of the domain of the Oracle CEP server.
server-name	string	Specifies the name of the Oracle CEP server.
protocol	string	Specifies the JTA protocol. Default: t3.
host-address	string	The host name or IP address of the Oracle CEP server.
port	int	The netio port for the Oracle CEP server.

F.41.2 Attributes

The `weblogic-instances` server configuration element has no attributes.

F.41.3 Example

The following example shows how to use the `weblogic-instances` element in the Oracle CEP server configuration file:

```
<weblogic-jta-gateway>
    <name>myJTAGateway</name>
    <timeout-seconds>90</timeout-seconds>
    <weblogic-instances>
        <weblogic-instance>
            <domain-name>ocep_domain</domain-name>
            <server-name>fxserver</server-name>
            <protocol>t3</protocol>
            <host-address>ariel</host-address>
            <port>9002</port>
        </weblogic-instance>
    </weblogic-instances>
</weblogic-jta-gateway>
```

F.42 weblogic-jta-gateway

Use this element to configure the attributes for the singleton Oracle CEP server client JTA gateway service.

F.42.1 Child Elements

The `weblogic-jta-gateway` server configuration element supports the following child elements:

- `name`

- `timeout-seconds`

- `weblogic-instances`

F.42.2 Attributes

The `weblogic-jta-gateway` server configuration element has no attributes.

F.42.3 Example

The following example shows how to use the `weblogic-jta-gateway` element in the Oracle CEP server configuration file:

```
<weblogic-jta-gateway>
    <name>myJTAGateway</name>
    <timeout-seconds>90</timeout-seconds>
    <weblogic-instances>
        <weblogic-instance>
            <domain-name>ocep_domain</domain-name>
            <server-name>fxserver</server-name>
            <protocol>t3</protocol>
            <host-address>ariel</host-address>
            <port>9002</port>
        </weblogic-instance>
    </weblogic-instances>
</weblogic-jta-gateway>
```

In the example, the weblogic-jta-gateway element's unique identifier is `myJTAGateway`.

F.43 weblogic-rmi-client

Use this element to configure the attributes for the singleton Oracle CEP server RMI client.

F.43.1 Child Elements

The `weblogic-rmi-client` server configuration element supports the child elements that Table F–30 lists.

Table F–30   Child Elements of: weblogic-rmi-client

XML Tag	Type	Description
name	string	The name of this `weblogic-rmi-client` element. For more information, see name.
netio-name	string	Specifies the name of the `netio-client` element to use. For more information, see netio-client.
secure-netio-name	string	Specifies the name of the `netio-client` element configured for SSL. For more information, see netio-client.

F.43.2 Attributes

The `weblogic-rmi-client` server configuration element has no attributes.

F.43.3 Example

The following example shows how to use the `weblogic-rmi-client` element in the Oracle CEP server configuration file:

```
<netio-client>
    <name>netio</name>
    <provider-type>NIO</provider-type>
</netio-client>

<netio-client>
    <name>netiossl</name>
    <provider-type>NIO</provider-type>
    <ssl-config-bean-name>sslConfig</ssl-config-bean-name>
</netio-client>

<weblogic-rmi-client>
    <name>wlclient</name>
    <netio-name>netio</netio-name>
    <secure-netio-name>netiossl</secure-netio-name>
</weblogic-rmi-client>
```

In the example, the weblogic-rmi-client element's unique identifier is `wlclient`.

F.44 work-manager

Use this element to configure a work manager on the Oracle CEP server.

F.44.1 Child Elements

The `work-manager` server configuration element supports the child elements that Table F–31 lists.

Table F–31   Child Elements of: work-manager

XML Tag	Type	Description
name	string	The name of this `work-manager` element. For more information, see name.
min-threads-constraint	int	The minimum threads constraint this work manager should use Default: -1.
fairshare	int	The fairshare value this work manager should use Default: -1.
max-threads-constraint	int	The maximum threads constraint this work manager should use Default: -1.

F.44.2 Attributes

The `work-manager` server configuration element has no attributes.

F.44.3 Example

The following example shows how to use the `work-manager` element in the Oracle CEP server configuration file:

```
<work-manager>
    <name>WM</name>
    <fairshare>5</fairshare>
```

```
        <min-threads-constraint>1</min-threads-constraint>
        <max-threads-constraint>4</max-threads-constraint>
    </work-manager>
```

In the example, the `work-manager` element's unique identifier is `WM`.

F.45 xa-params

Use this element to specify distributed transaction-related `data-source` parameters.

F.45.1 Child Elements

The `xa-params` server configuration element supports the child elements that Table F–32 lists.

Table F–32   Child Elements of: xa-params

XML Tag	Type	Description
`keep-xa-conn-till-tx-complete`	boolean	Enables the server to associate the same XA database connection from the connection pool with a global transaction until the transaction completes. Only applies to connection pools that use an XA driver. Use this setting to work around specific problems with JDBC XA drivers. Default: `true`.
`xa-transaction-timeout`	int	The number of seconds to set as the transaction branch timeout. If set, this value is passed as the transaction timeout value in the `XAResource.setTransactionTimeout()` call on the XA resource manager, typically the JDBC driver. When this value is set to 0, the Transaction Manager passes the global server transaction timeout in seconds in the method. If set, this value should be greater than or equal to the global server transaction timeout. Note: You must enable `xa-set-transaction-timeout` to enable setting the transaction branch timeout. Default: 0.
`rollback-local-tx-upon-conn-close`	boolean	Enables the server to call `rollback()` on the connection before returning the connection to the connection pool. Enabling this attribute will have a performance impact as the rollback call requires communication with the database server. Default: `false`.
`xa-retry-duration-seconds`	int	Determines the duration in seconds for which the transaction manager will perform recover operations on the resource. A value of zero indicates that no retries will be performed. Default: 60.
`xa-set-transaction-timeout`	boolean	Enables the server to set a transaction branch timeout based on the value for `xa-transaction-timeout`. When enabled, the Transaction Manager calls `XAResource.setTransactionTimeout()` before calling XAResource.start, and passes either the XA Transaction Timeout value or the global transaction timeout. You may want to set a transaction branch timeout if you have long-running transactions that exceed the default timeout value on the XA resource. Default: `false`.

Table F–32   (Cont.) Child Elements of: xa-params

XML Tag	Type	Description
keep-logical-conn-open-on-release	boolean	Enables the server to keep the logical JDBC connection open for a global transaction when the physical XA connection is returned to the connection pool. Select this option if the XA driver used to create database connections or the DBMS requires that a logical JDBC connection be kept open while transaction processing continues (although the physical XA connection can be returned to the connection pool). Only applies to data sources that use an XA driver. Use this setting to work around specific problems with JDBC XA drivers. Default: `false`.
resource-health-monitoring	boolean	Enables JTA resource health monitoring for an XA data source. When enabled, if an XA resource fails to respond to an XA call within the period specified in MaxXACallMillis, the server marks the data source as unhealthy and blocks any further calls to the resource. This property applies to XA data sources only, and is ignored for data sources that use a non-XA driver. Default: `true`.
new-xa-conn-for-commit	boolean	Specifies that a dedicated XA connection is used for commit and rollback processing for a global transaction. Only applies to data sources that use an XA driver. Use this setting to work around specific problems with JDBC XA drivers. Default: `false`.
xa-end-only-once	boolean	Specifies that XAResource.end() is called only once for each pending `XAResource.start()`. This option prevents the XA driver from calling `XAResource.end(TMSUSPEND)` and `XAResource.end(TMSUCCESS)` successively. Only applies to data sources that use an XA driver. Use this setting to work around specific problems with JDBC XA drivers. Default: `false`.
xa-retry-interval-seconds	int	The number of seconds between XA retry operations if XARetryDurationSeconds is set to a positive value. Default: 60.
recover-only-once	boolean	Specifies that the transaction manager calls recover on the resource only once. Only applies to data sources that use an XA driver. Use this setting to work around specific problems with JDBC XA drivers. Default: `false`.
need-tx-ctx-on-close	boolean	Specifies whether the XA driver requires a distributed transaction context when closing various JDBC objects (result sets, statements, connections, and so forth). Only applies to connection pools that use an XA driver. When enabled, SQL exceptions that are thrown while closing the JDBC objects without a transaction context will be suppressed. Use this setting to work around specific problems with JDBC XA drivers. Default: `false`.

F.45.2 Attributes

The `xa-params` server configuration element has no attributes.

F.45.3 Example

The following example shows how to use the `xa-params` element in the Oracle CEP server configuration file:

```
<data-source>
    <name>orads</name>
    <xa-params>
        <keep-xa-conn-till-tx-complete>true</keep-xa-conn-till-tx-complete>
    </xa-params>
```

```
<driver-params>
    <url>jdbc:oracle:thin:@localhost:1521:ce102</url>
    <driver-name>oracle.jdbc.OracleDriver</driver-name>
    <properties>
        <element>
            <name>user</name>
            <value>wlevs</value>
        </element>
        <element>
            <name>password</name>
            <value>wlevs</value>
        </element>
    </properties>
</driver-params>
<connection-pool-params>
    <initial-capacity>5</initial-capacity>
    <max-capacity>10</max-capacity>
    <test-table-name>SQL SELECT 1 FROM DUAL</test-table-name>
    <test-frequency-seconds>5</test-frequency-seconds>
</connection-pool-params>
<data-source-params>
    <jndi-names>
        <element>orads</element>
    </jndi-names>
    <global-transactions-protocol>None</global-transactions-protocol>
</data-source-params>
</data-source>
```

G

Oracle CEP Metadata Annotation Reference

This section contains information on the following subjects:

G.1 Overview of Oracle CEP Metadata Annotations

Oracle CEP metadata annotations are used to access the configuration of an Oracle CEP component. Oracle CEP offers the following annotations:

- Section G.1.1, "Adapter Lifecycle Annotations"

- Section G.1.2, "OSGi Service Reference Annotations"

- Section G.1.3, "Resource Access Annotations"

For more information, see:

- Section 1.1.9, "Oracle CEP Application Lifecycle"

- Section 1.6, "Configuring Oracle CEP Resource Access"

G.1.1 Adapter Lifecycle Annotations

You use the following 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

G.1.2 OSGi Service Reference Annotations

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

G.1.3 Resource Access Annotations

Use the following annotations to configure resource access at design time and the corresponding deployment XML to override this configuration at deploy time:

- `javax.annotation.Resource`

For more information, see Section 1.6, "Configuring Oracle CEP Resource Access".

G.2 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 representation 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.

G.2.1 Example

Example G–1 shows how to use the `@Activate` annotation in the adapter component of the HelloWorld example; only relevant code is shown:

Example G–1   @Activate Annotation

```
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. Example G–2 shows this XSD file also specifies the fully qualified name
of the resulting Java configuration object, as shown in bold:

Example G–2   HelloWorldAdapterConfig

```
<?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="wlev
s_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="channel" 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`
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.

G.3 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.

G.3.1 Example

Example G–3, from the adapter component of the HelloWorld example, shows how to use the `@Prepare` annotation; only relevant code is shown:

Example G–3   @Prepare Annotation

```
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
Appendix G.2, "com.bea.wlevs.management.Activate" 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.

G.4 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.

G.4.1 Example

Example G–4, sample code from the adapter component of the HelloWorld example, shows how to use the @Rollback annotation; only relevant code is shown:

Example G–4   @Rollback Annotation

```
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 Appendix G.2, "com.bea.wlevs.management.Activate" 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.

G.5 com.bea.wlevs.util.Service

Target: Method

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

G.5.1 Attributes

Table G–1 describes the attributes of the com.bea.wlevs.util.Service JWS annotation.

Table G–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 Default value is ServiceCardinality.C1__1.	enum	No.

Table G–1   (Cont.) Attributes of the com.bea.wlevs.util.Service JWS Annotation Tag

Name	Description	Data Type	Required?
contextClassloader	Valid values for this attribute are: ■ ServiceClassloader.CLIENT ■ ServiceClassloader.SERVICE_PROVIDER ■ ServiceClassloader.UNMANAGED Default value is ServiceClassloader.CLIENT.	enum	No.
timeout	Timeout for service 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.

G.5.2 Example

Example G–5 shows how to use the @Service annotation.

Example G–5   @Service Annotation

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

H

Oracle CEP IDE for Eclipse Tutorial

This appendix provides a simple tutorial that illustrates the basics of creating, deploying, and debugging a simple Oracle CEP application.

This tutorial includes the following steps:

- Appendix H.1, "Before You Begin"
- Appendix H.2, "Step 1: Create an Oracle CEP Definition"
- Appendix H.3, "Step 2: Create an Oracle CEP Application"
- Appendix H.4, "Step 3: Start the Oracle CEP Server and Deploy the Project"
- Appendix H.5, "Step 4: Change Code and Redeploy"
- Appendix H.6, "Step 5: Debug the Deployed Application"
- Appendix H.7, "Next Steps"

H.1 Before You Begin

This tutorial assumes that you have installed an Oracle CEP server and Oracle CEP IDE for Eclipse on the local computer. For more information, see:
http://www.oracle.com/technologies/soa/complex-event-processing.html.

Before beginning this tutorial it is a good idea to switch to the Java or the Java EE perspective: both of which are good starting points for using Oracle CEP IDE for Eclipse to develop Oracle CEP applications. Although Oracle CEP applications are not Java EE applications, many of the same views are useful between the two.

To change the perspective, select **Window > Open Perspective** menu, or use the perspective shortcut menu as shown in Figure H–1.

Figure H–1   Perspective Shortcut Menu
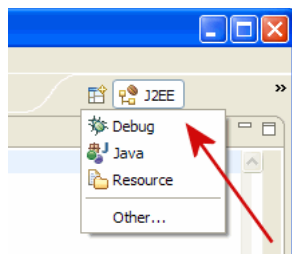
If you are in the Java perspective, you will want to open the Servers view by selecting **Window > Show View > Servers**.

Your Oracle CEP IDE for Eclipse should look like Figure H–2.

Figure H–2 Oracle CEP IDE for Eclipse
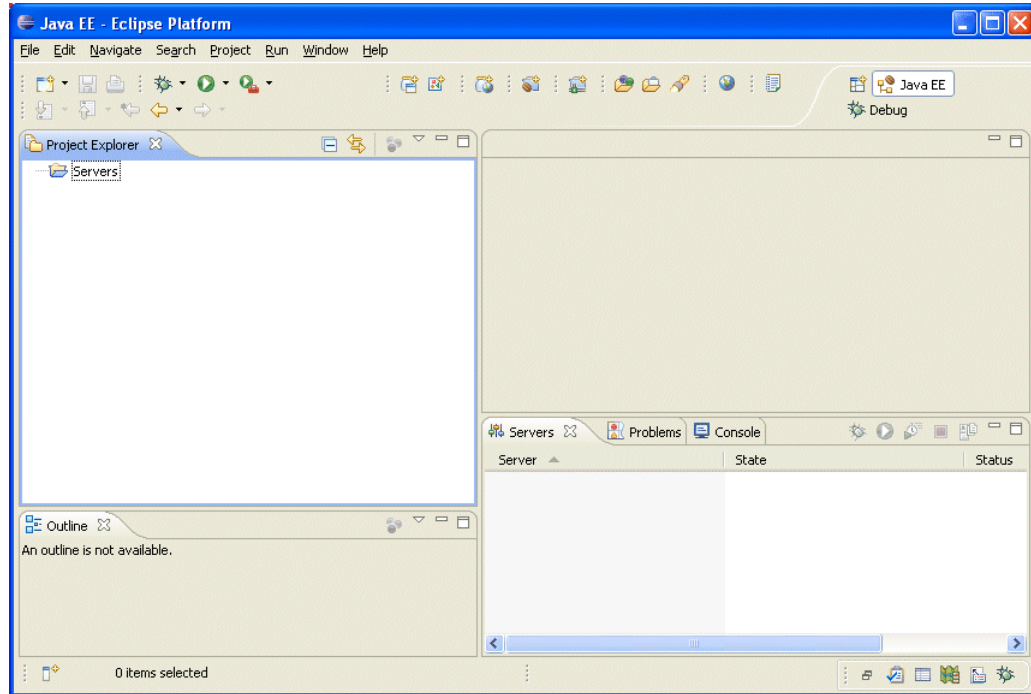
H.2 Step 1: Create an Oracle CEP Definition

The first step is to tell the Oracle CEP IDE for Eclipse where your Oracle CEP server is installed on the local machine. To do this, follow the steps in the Section 4.2.2, "How to Create an Oracle CEP Server and Server Runtime".

For this tutorial, enable automatic publishing on the server by selecting Use Workspace Default for the Automatic Publishing options on the second wizard page as Figure H–3 shows.

Figure H–3   Configuring the Oracle CEP Server for Automatic Publishing
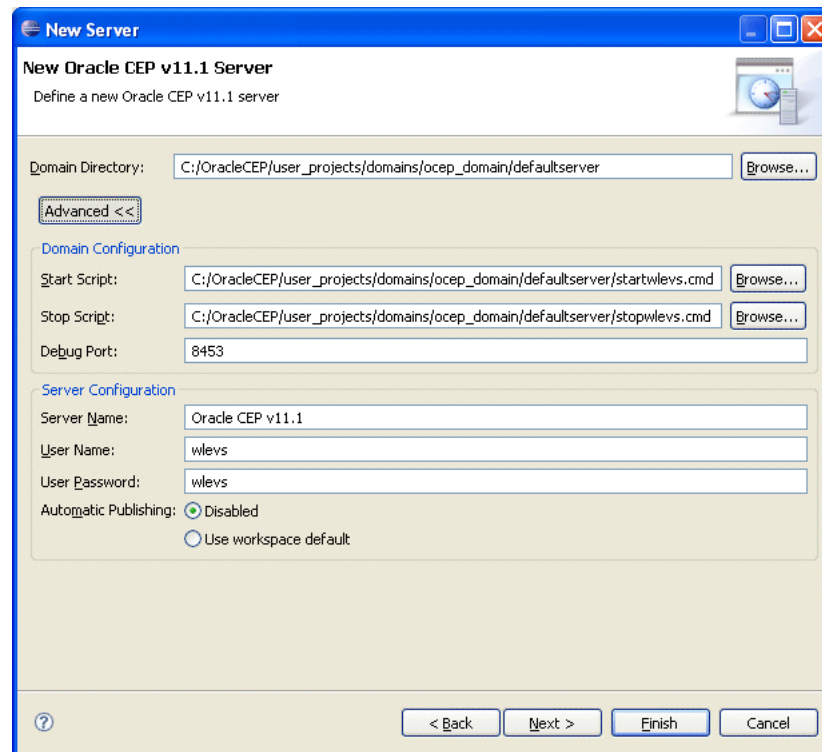
When finished, your Servers view should look as Figure H–4 shows.

Figure H–4   Oracle CEP IDE for Eclipse Server View
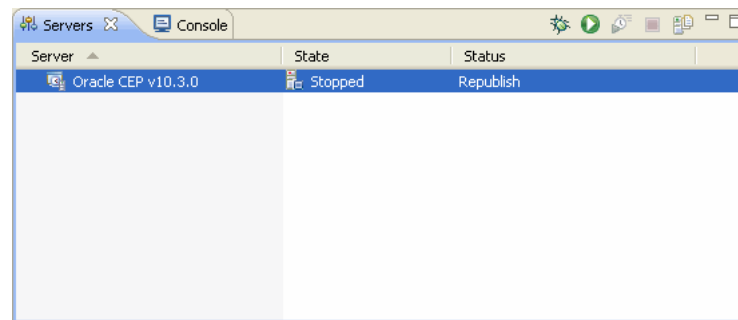
H.3 Step 2: Create an Oracle CEP Application

Next, create an Oracle CEP application. The basic the steps are described in
Section 4.2.2, "How to Create an Oracle CEP Server and Server Runtime".

In this tutorial, select the **File > New > Other**, expand the **Oracle CEP** option and
select **Oracle CEP Application Project**.

In the New Oracle CEP Application Project dialog, enter a project name of `Hello` and
select the `Oracle CEP v10.3.0` the Target Runtime (created in Appendix H.2, "Step
1: Create an Oracle CEP Definition") as Figure H.5 shows.

Figure H–5   New Oracle CEP Application Project Dialog
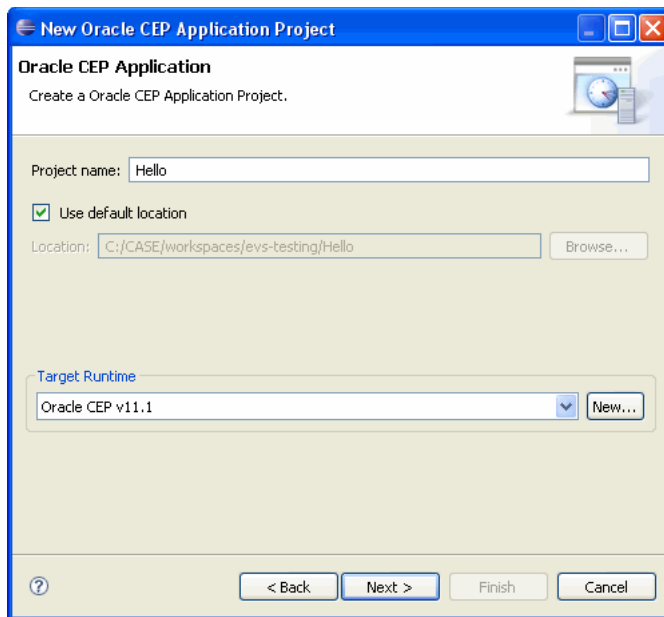
Click **Next**. Leave the Bundle Properties at their defaults and click **Next** again. The Templates dialog appears as Figure H–6 shows.

Figure H–6   Templates Dialog
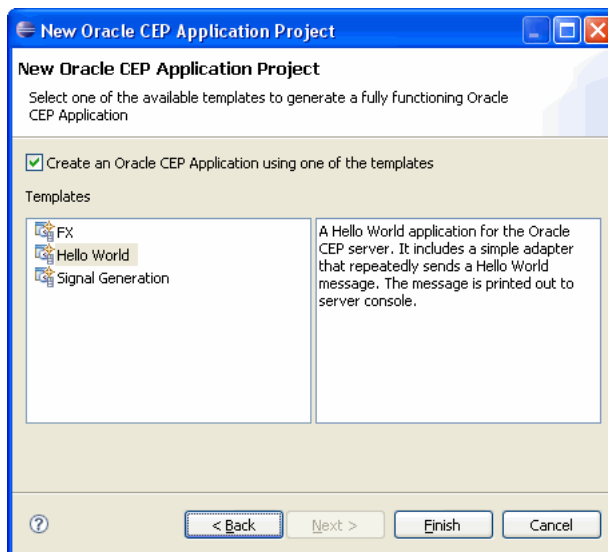
This page allows you to select what initial application files should be placed in the project. Check **Create an Oracle CEP Application using one of these templates** and select the **Hello World** template.

Click **Finish**.

The Project Explorer should contain the files shown in Figure H–7.

Figure H–7   Project Explorer
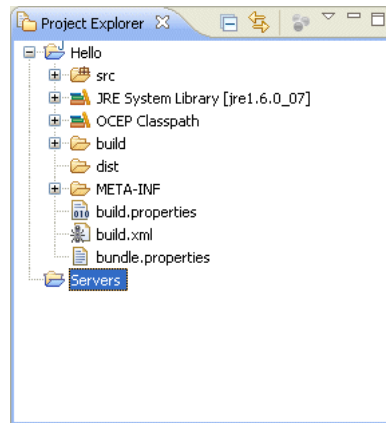
H.4 Step 3: Start the Oracle CEP Server and Deploy the Project

Now that the project is created and includes example content, it's time to start the server and deploy the project.

To start the server, click the **Debug** button on the Servers view as shown in Figure H–8. The debug icon looks like a bug on the view toolbar.

Figure H–8   Starting the Oracle CEP Server
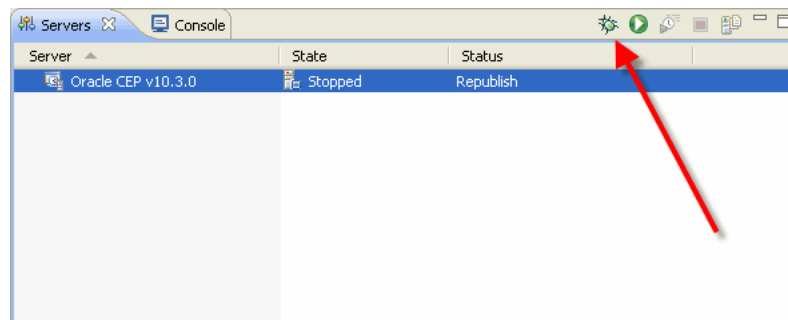
In the Console view, some Oracle CEP server log messages will scroll by, followed by the server indicating that it has started, such as:

```
<Jan 22, 2009 4:59:41 PM EST> <Notice> <Server> <BEA-2046000> <Server STARTED>
```

Next, deploy the project to the server. In the Servers view, right-click the server and select **Add/Remove Projects**. The Add and Remove Projects dialog appears as shown in Figure H–9.

Figure H–9 Add and Remove Projects Dialog
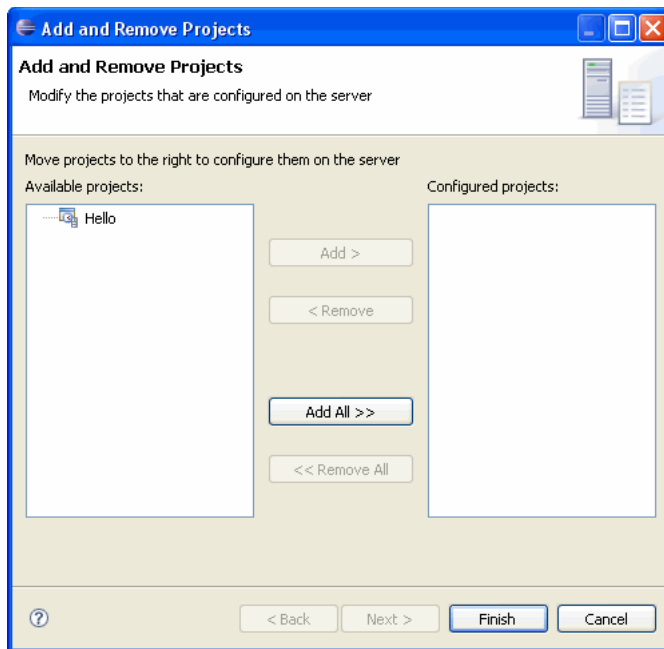
Select the **Hello** project from the Available projects list on the left and click **Add** to move the project to the Configured projects list on the right.

Click **Finish**. The Oracle CEP IDE for Eclipse deploys the project to the server.

In the Console view, the Oracle CEP server indicates that the application is deployed, such as:

```
<Jan 23, 2009 9:24:13 AM EST> <Notice> <Spring> <BEA-2047000> <The application context for
"com.bea.wlevs.monitor" was deployed successfully>
<Jan 23, 2009 9:24:17 AM EST> <Notice> <Spring> <BEA-2047000> <The application context for
"com.bea.wlevs.dataservices" was deployed successfully>
<Jan 23, 2009 9:26:06 AM EST> <Notice> <Server> <BEA-2045000> <The application bundle "Hello"
was deployed successfully to file:/C:/OracleCEP/user_projects/domains/ocep_
domain/defaultserver/applications/Hello/Hello.jar with version 1232720766696>
<Jan 23, 2009 9:26:06 AM EST> <Notice> <Server> <BEA-2045000> <The application bundle "Hello"
was deployed successfully to file [C:\OracleCEP\user_projects\domains\ocep_
domain\defaultserver\applications\Hello\Hello.jar] with version 1232720766696>
<Jan 23, 2009 9:26:22 AM EST> <Notice> <Spring> <BEA-2047000> <The application context for
"Hello" was deployed successfully>
```

Because the application is now deployed and running, you should see messages from your application being logged to the Console view

```
Message: HelloWorld - the current time is:9:26:22 AM
Message: HelloWorld - the current time is:9:26:23 AM
Message: HelloWorld - the current time is:9:26:23 AM
...
```

H.5 Step 4: Change Code and Redeploy

Next, we will change some code and redeploy the project to demonstrate the iterative development cycle.

Open the EPN (see Section 5.1, "Opening the EPN Editor").

In the EPN Editor, right click on **helloworldAdapter** and select **Go to Java Source** as shown in Figure H–10.

Figure H–10   Editing Java Source for an Adapter
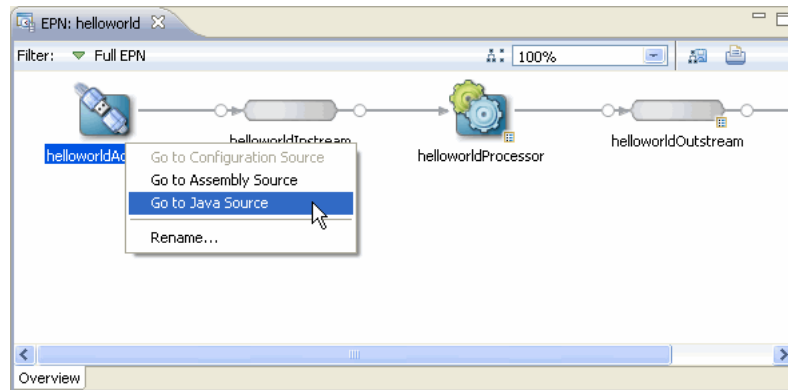
The Java editor opens as Figure H–11 shows.

Figure H–11   Java Editor for helloworldAdapter
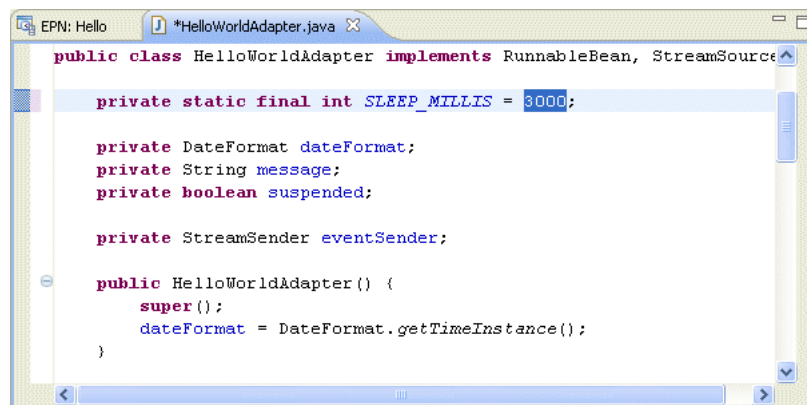
Using the Java editor, change the value of the SLEEP_MILLIS field to 3000 so that events only fire every 3 seconds. Select **File > Save**.

Because the default setting is to automatically redeploy projects, no further action is required. Oracle CEP IDE for Eclipse will redeploy the project after a few seconds and the rate of HelloWorld messages in the Console view will slow down to every 3 seconds as shown below.

```
Message: HelloWorld - the current time is:9:37:41 AM
Message: HelloWorld - the current time is:9:37:41 AM
Message: HelloWorld - the current time is:9:37:41 AM
<Jan 23, 2009 9:37:42 AM EST> <Notice> <Spring> <BEA-2047001> <The application context
"Hello" was undeployed successfully>
<Jan 23, 2009 9:37:42 AM EST> <Notice> <Server> <BEA-2045000> <The application bundle "Hello"
was deployed successfully to file:/C:/OracleCEP/user_projects/domains/ocep_
domain/defaultserver/applications/Hello/Hello.jar with version 1232721461898>
<Jan 23, 2009 9:37:43 AM EST> <Notice> <Spring> <BEA-2047000> <The application context for
"Hello" was deployed successfully>
Message: HelloWorld - the current time is:9:37:43 AM
Message: HelloWorld - the current time is:9:37:46 AM
```

If automatic publishing is disabled, to republish the project, right-click on the server in the Servers view and select **Publish** as Figure H–12 shows.

Figure H–12   Manually Publishing a Project
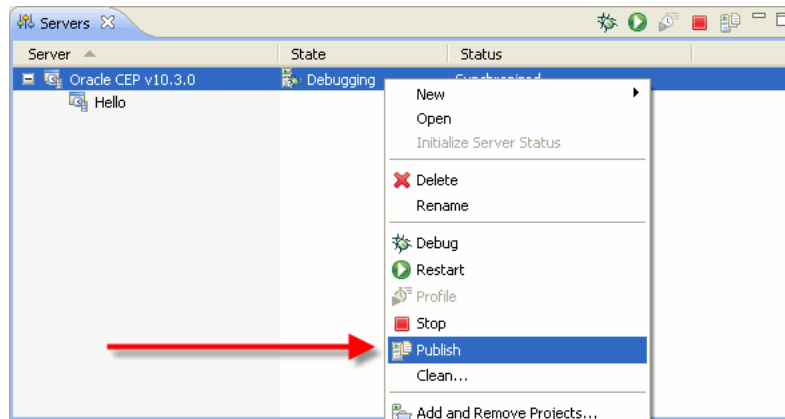
H.6 Step 5: Debug the Deployed Application

Next, we will debug some code now that the application is up and running.

Open the EPN (see Section 5.1, "Opening the EPN Editor").

In the EPN Editor, right click on **HelloWorldBean** and select **Go to Java Source** as shown in Figure H–13.

Figure H–13   Java Editor for HelloWorldBean
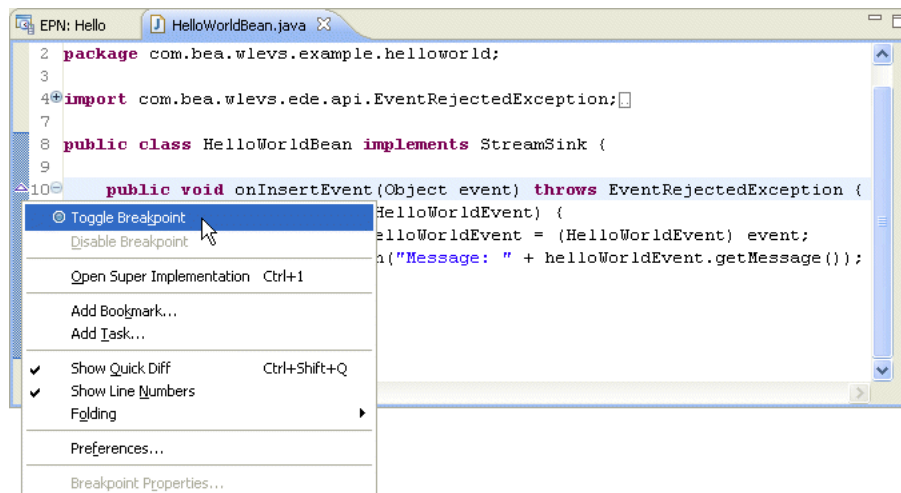
Right-click in the gutter on line 10 (to the left of the line number), and select **Toggle Breakpoint**.

The next time this `StreamSink` receives a message, it will stop at line 10 where the breakpoint was set and enter the Debug view as Figure H–14 shows.

Figure H–14   HelloWorldBean in the Debugger
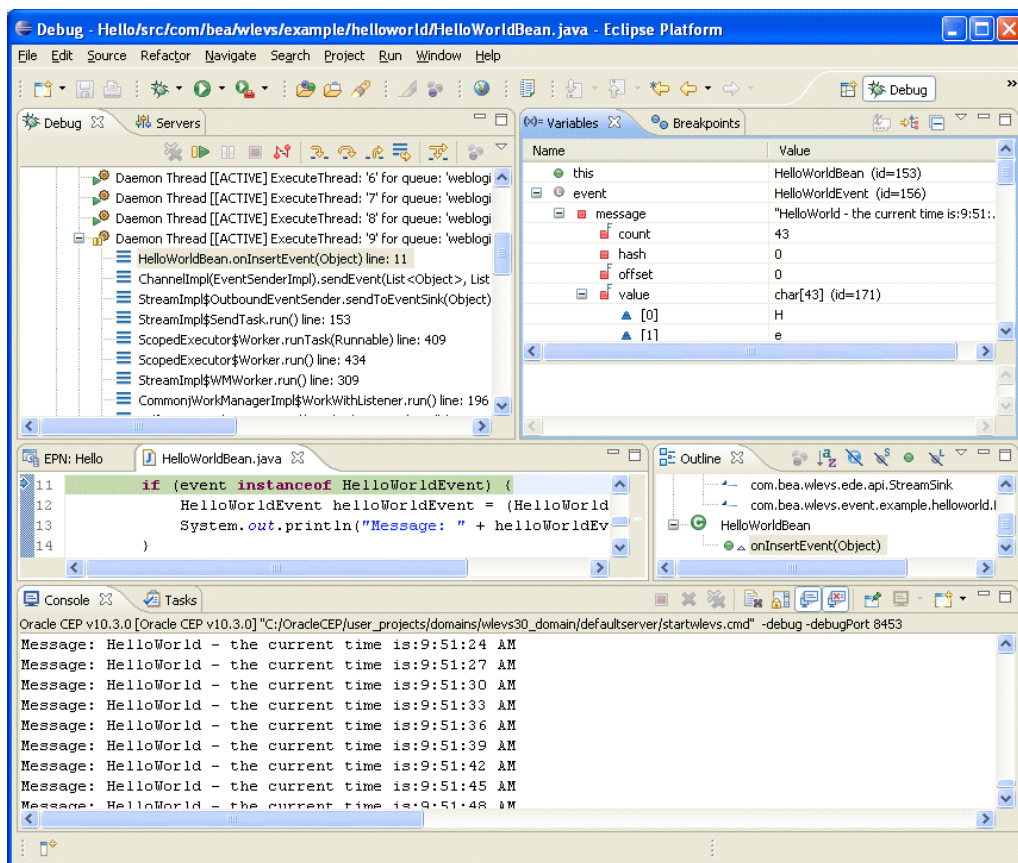
In the Variables view, you can expand the `event` variable and view the values of its attributes.

H.7 Next Steps

Examine the Oracle CEP examples available in the Oracle CEP server installer.

For more information, see "Oracle CEP Samples" in the *Oracle CEP Getting Started*

Index

Z