

## **Oracle Fusion Middleware**

Developer's Guide for Oracle Event Processing

11g Release 1 (11.1.1.9)

**E14301-11**

February 2015

Documentation for developers that describes how to build Oracle Event Processing scalable applications to process streaming events.

Copyright © 2007, 2015, Oracle and/or its affiliates. All rights reserved.

Primary Author: Oracle Corporation

This software and related documentation are provided under a license agreement containing restrictions on use and disclosure and are protected by intellectual property laws. Except as expressly permitted in your license agreement or allowed by law, you may not use, copy, reproduce, translate, broadcast, modify, license, transmit, distribute, exhibit, perform, publish, or display any part, in any form, or by any means. Reverse engineering, disassembly, or decompilation of this software, unless required by law for interoperability, is prohibited.

The information contained herein is subject to change without notice and is not warranted to be error-free. If you find any errors, please report them to us in writing.

If this is software or related documentation that is delivered to the U.S. Government or anyone licensing it on behalf of the U.S. Government, then the following notice is applicable:

U.S. GOVERNMENT END USERS: Oracle programs, including any operating system, integrated software, any programs installed on the hardware, and/or documentation, delivered to U.S. Government end users are "commercial computer software" pursuant to the applicable Federal Acquisition Regulation and agency-specific supplemental regulations. As such, use, duplication, disclosure, modification, and adaptation of the programs, including any operating system, integrated software, any programs installed on the hardware, and/or documentation, shall be subject to license terms and license restrictions applicable to the programs. No other rights are granted to the U.S. Government.

This software or hardware is developed for general use in a variety of information management applications. It is not developed or intended for use in any inherently dangerous applications, including applications that may create a risk of personal injury. If you use this software or hardware in dangerous applications, then you shall be responsible to take all appropriate fail-safe, backup, redundancy, and other measures to ensure its safe use. Oracle Corporation and its affiliates disclaim any liability for any damages caused by use of this software or hardware in dangerous applications.

Oracle and Java are registered trademarks of Oracle and/or its affiliates. Other names may be trademarks of their respective owners.

Intel and Intel Xeon are trademarks or registered trademarks of Intel Corporation. All SPARC trademarks are used under license and are trademarks or registered trademarks of SPARC International, Inc. AMD, Opteron, the AMD logo, and the AMD Opteron logo are trademarks or registered trademarks of Advanced Micro Devices. UNIX is a registered trademark of The Open Group.

This software or hardware and documentation may provide access to or information about content, products, and services from third parties. Oracle Corporation and its affiliates are not responsible for and expressly disclaim all warranties of any kind with respect to third-party content, products, and services unless otherwise set forth in an applicable agreement between you and Oracle. Oracle Corporation and its affiliates will not be responsible for any loss, costs, or damages incurred due to your access to or use of third-party content, products, or services, except as set forth in an applicable agreement between you and Oracle.

---

---

# Contents

<b>Preface</b> .....	li
Audience .....	li
Documentation Accessibility .....	li
Related Documents .....	li
Conventions .....	lii
<b>What's New in This Guide</b> .....	liii
<b>Part I Getting Started with Creating Oracle Event Processing Applications</b>	
<b>1 Overview of Creating Oracle Event Processing Applications</b>	
<b>Oracle Event Processing Application Programming Model</b> .....	1-1
Key Concepts Underlying Oracle Event Processing Applications .....	1-2
Component Roles in an Event Processing Network .....	1-2
Tools and Supporting Technologies for Developing Applications .....	1-3
<b>How an Oracle Event Processing Application Works</b> .....	1-4
<b>Overview of Events, Streams and Relations</b> .....	1-8
<b>Overview of Application Configuration</b> .....	1-9
Overview of EPN Assembly Files .....	1-10
Nesting Stages in an EPN Assembly File .....	1-11
Referencing Foreign Stages in an EPN Assembly File .....	1-12
Overview of Component Configuration Files .....	1-12
Accessing Component and Server Configuration Using the ConfigurationPropertyPlaceholderConfigurer Class .....	1-13
Configuring Oracle Event Processing Resource Access .....	1-14
Static Resource Injection .....	1-15
Static Resource Names .....	1-15
Dynamic Resource Names .....	1-16
Dynamic Resource Injection .....	1-16
Dynamic Resource Lookup Using JNDI .....	1-17
Understanding Resource Name Resolution .....	1-17
<b>Oracle Event Processing APIs</b> .....	1-18
<b>Packaging an Application</b> .....	1-20
<b>Oracle Event Processing Application Lifecycle</b> .....	1-20

## 2 Oracle Event Processing Samples

<b>Overview of the Samples Provided in the Distribution Kit</b> .....	2-1
Ready-to-Run Samples .....	2-2
Sample Source.....	2-2
<b>Installing the Default ocep_domain and Samples</b> .....	2-3
<b>Using Oracle Event Processing Visualizer With the Samples</b> .....	2-3
<b>Increasing the Performance of the Samples</b> .....	2-4
<b>HelloWorld Example</b> .....	2-4
Running the HelloWorld Example from the helloworld Domain .....	2-5
Building and Deploying the HelloWorld Example from the Source Directory .....	2-6
Description of the Ant Targets to Build Hello World .....	2-6
Implementation of the HelloWorld Example.....	2-7
<b>Oracle Continuous Query Language (Oracle CQL) Example</b> .....	2-8
Running the CQL Example.....	2-9
Building and Deploying the CQL Example .....	2-11
Description of the Ant Targets to Build the CQL Example .....	2-11
Implementation of the CQL Example .....	2-12
Creating the Missing Event Query .....	2-12
Creating the Moving Average Query .....	2-36
<b>Oracle Spatial Example</b> .....	2-71
Running the Oracle Spatial Example .....	2-72
Building and Deploying the Oracle Spatial Example .....	2-76
Description of the Ant Targets to Build the Oracle Spatial Example .....	2-77
Implementation of the Oracle Spatial Example .....	2-77
<b>Foreign Exchange (FX) Example</b> .....	2-78
Running the Foreign Exchange Example .....	2-79
Building and Deploying the Foreign Exchange Example from the Source Directory .....	2-81
Description of the Ant Targets to Build FX .....	2-82
Implementation of the FX Example.....	2-82
<b>Signal Generation Example</b> .....	2-83
Running the Signal Generation Example .....	2-84
Building and Deploying the Signal Generation Example from the Source Directory .....	2-87
Description of the Ant Targets to Build Signal Generation .....	2-87
Implementation of the Signal Generation Example .....	2-88
<b>Event Record and Playback Example</b> .....	2-89
Running the Event Record/Playback Example .....	2-90
Building and Deploying the Event Record/Playback Example from the Source Directory .....	2-96
Description of the Ant Targets to Build the Record and Playback Example.....	2-97
Implementation of the Record and Playback Example .....	2-98

## 3 Getting Started with Developing Oracle Event Processing Applications

<b>Creating an Oracle Event Processing Application</b> .....	3-1
<b>Setting Your Development Environment</b> .....	3-3
How to Set Your Development Environment on Windows .....	3-4
How to Set Your Development Environment on UNIX.....	3-5
<b>Using an IDE to Develop Applications</b> .....	3-6
<b>Testing Applications</b> .....	3-6

## Part II Oracle Event Processing IDE for Eclipse

### 4 Overview of the Oracle Event Processing IDE for Eclipse

Overview of Oracle Event Processing IDE for Eclipse .....	4-1
Features.....	4-1
JDK Requirements.....	4-2
Default Oracle Event Processing Domain ocep_domain and Development .....	4-2
Installing the Latest Oracle Event Processing IDE for Eclipse.....	4-2
Installing the Oracle Event Processing IDE for Eclipse Distributed With Oracle Event Processing .....	4-7
Configuring Eclipse .....	4-11

### 5 Oracle Event Processing IDE for Eclipse Projects

Oracle Event Processing Project Overview .....	5-1
Creating Oracle Event Processing Projects.....	5-2
How to Create an Oracle Event Processing Project.....	5-3
Creating EPN Assembly Files .....	5-6
How to Create a New EPN Assembly File Using Oracle Event Processing IDE for Eclipse ..	5-7
Creating Component Configuration Files .....	5-8
How to Create a New Component Configuration File Using Oracle Event Processing IDE for Eclipse	5-9
Exporting Oracle Event Processing Projects .....	5-10
How to Export an Oracle Event Processing Project .....	5-11
Upgrading Projects.....	5-13
How to Upgrade Projects from Oracle Event Processing 2.1 to 10.3.....	5-13
How to Upgrade Projects from Oracle Event Processing 10.3 to 11g Release 1 (11.1.1).....	5-18
Managing Libraries and Other Non-Class Files in Oracle Event Processing Projects .....	5-25
How to Add a Standard JAR File to an Oracle Event Processing Project.....	5-26
How to Add an OSGi Bundle to an Oracle Event Processing Project .....	5-31
How to Add a Property File to an Oracle Event Processing Project.....	5-32
How to Export a Package.....	5-34
How to Import a Package .....	5-36
Configuring Oracle Event Processing IDE for Eclipse Preferences .....	5-39
How to Configure Application Library Path Preferences.....	5-39
How to Configure Problem Severity Preferences.....	5-39

### 6 Oracle Event Processing IDE for Eclipse and Oracle Event Processing Servers

Oracle Event Processing Server Overview .....	6-1
Creating Oracle Event Processing Servers.....	6-3
How to Create a Local Oracle Event Processing Server and Server Runtime.....	6-3
How to Create a Remote Oracle Event Processing Server and Server Runtime.....	6-10
How to Create an Oracle Event Processing Server Runtime .....	6-17
Managing Oracle Event Processing Servers.....	6-20
How to Start a Local Oracle Event Processing Server .....	6-20
How to Stop a Local Oracle Event Processing Server.....	6-21

How to Attach to an Existing Local Oracle Event Processing Server Instance .....	6-22
How to Attach to an Existing Remote Oracle Event Processing Server Instance .....	6-22
How to Detach From an Existing Oracle Event Processing Server Instance .....	6-23
How to Deploy an Application to an Oracle Event Processing Server .....	6-24
How to Configure Connection and Control Settings for Oracle Event Processing Server ..	6-27
How to Configure Domain (Runtime) Settings for Oracle Event Processing Server .....	6-29
How to Start the Oracle Event Processing Visualizer from Oracle Event Processing IDE for Eclipse	6-31
<b>Debugging an Oracle Event Processing Application Running on an Oracle Event Processing Server .....</b>	<b>6-33</b>
How to Debug an Oracle Event Processing Application Running on an Oracle Event Processing Server	6-34

## 7 Oracle Event Processing IDE for Eclipse and the Event Processing Network

<b>Opening the EPN Editor .....</b>	<b>7-1</b>
How to Open the EPN Editor from a Project Folder .....	7-1
How to Open the EPN Editor from a Context or Configuration File .....	7-3
<b>EPN Editor Overview .....</b>	<b>7-4</b>
Flow Representation .....	7-4
Filtering .....	7-5
Zooming .....	7-6
Layout .....	7-6
Showing and Hiding Unconnected Beans .....	7-6
Printing and Exporting to an Image .....	7-7
Configuration Badging .....	7-7
Link Specification Location Indicator .....	7-8
Nested Stages .....	7-9
Event Type Repository Editor .....	7-10
<b>Navigating the EPN Editor .....</b>	<b>7-11</b>
Moving the Canvas .....	7-11
Shortcuts to Component Configuration and EPN Assembly Files .....	7-11
Hyperlinking .....	7-12
Hyperlinking in Component Configuration and EPN Assembly Files .....	7-12
Hyperlinking in Oracle CQL Statements .....	7-13
Context Menus .....	7-14
Browsing Oracle Event Processing Types .....	7-15
How to Browse Oracle Event Processing Types .....	7-15
<b>Using the EPN Editor .....</b>	<b>7-18</b>
Creating Nodes .....	7-18
How to Create a Basic Node .....	7-19
How to Create an Adapter Node .....	7-21
How to Create a Processor Node .....	7-26
Connecting Nodes .....	7-28
How to Connect Nodes .....	7-28
Laying Out Nodes .....	7-30
Renaming Nodes .....	7-30
Deleting Nodes .....	7-30

## Part III Developing Oracle Event Processing Applications

### 8 Walkthrough: Assembling a Simple Application

Introduction to the Simple Application Walkthrough .....	8-1
Key Concepts in this Walkthrough .....	8-2
Before You Get Started .....	8-2
Create the Workspace and Project.....	8-2
Create an Event Type to Carry Event Data .....	8-7
Add an Input Adapter to Receive Event Data .....	8-11
Add a Channel to Convey Events.....	8-13
Create a Listener to Receive and Report Events.....	8-15
Set Up the Load Generator and Test.....	8-19
Add an Oracle CQL Processor to Filter Events.....	8-22
Summary: Simple Application Walkthrough.....	8-27

### 9 Defining and Using Event Types

Overview of Oracle Event Processing Event Types.....	9-1
Where Event Type Instances are Used.....	9-2
High-Level Process for Creating Event Types .....	9-2
Designing Event Types .....	9-2
Identifying the Structure of Event Data .....	9-3
Choosing a Data Type for an Event Type.....	9-4
Constraints on Design of Event Types .....	9-5
Constraints on Event Types for Use With the csvgen Adapter .....	9-5
Constraints on Event Types for Use With a Database Table Source .....	9-5
Mixing Use of Event Type Data Types .....	9-7
Creating Event Types.....	9-7
Creating an Oracle Event Processing Event Type as a JavaBean .....	9-7
How to Create an Oracle Event Processing Event Type as a JavaBean Using the Event Type Repository Editor 9-8	
How to Create an Oracle Event Processing Event Type as a JavaBean Manually .....	9-10
Using JavaBean Event Type Instances in Java Code.....	9-12
Using JavaBean Event Type Instances in Oracle CQL Code .....	9-12
Controlling Event Type Instantiation with an Event Type Builder Class .....	9-13
Implementing an Event Type Builder Class .....	9-13
Configuring an Event Type that Uses an Event Type Builder .....	9-14
Creating an Oracle Event Processing Event Type as a Tuple.....	9-14
Types for Properties in Tuple-Based Event Types.....	9-15
How to Create an Oracle Event Processing Event Type as a Tuple Using the Event Type Repository Editor 9-15	
How to Create an Oracle Event Processing Event Type as a Tuple Manually .....	9-17
Using a Tuple Event Type Instance in Java Code .....	9-17
Using a Tuple Event Type Instance in Oracle CQL Code.....	9-18
Creating an Oracle Event Processing Event Type as a java.util.Map .....	9-18
Types for Properties in java.util.Map-Based Event Types.....	9-18
How to Create an Oracle Event Processing Event Type as a java.util.Map .....	9-19

Using a Map Event Type Instance in Java Code .....	9-20
Using a Map Event Type Instance in Oracle CQL Code .....	9-20
<b>Accessing the Event Type Repository .....</b>	<b>9-20</b>
Using the EPN Assembly File .....	9-20
Using the Spring-DM @ServiceReference Annotation .....	9-21
Using the Oracle Event Processing @Service Annotation .....	9-21
<b>Sharing Event Types Between Application Bundles .....</b>	<b>9-22</b>

## 10 Connecting EPN Stages Using Channels

<b>Overview of Channel Configuration.....</b>	<b>10-1</b>
When to Use a Channel .....	10-2
Channels Representing Streams and Relations .....	10-3
Channels as Streams .....	10-3
Channels as Relations.....	10-3
System-Timestamped Channels.....	10-4
Application-Timestamped Channels .....	10-4
Controlling Which Queries Output to a Downstream Channel: selector .....	10-4
Batch Processing Channels .....	10-6
EventPartitioner Channels.....	10-6
Handling Faults in Channels.....	10-6
<b>Configuring a Channel.....</b>	<b>10-7</b>
How to Configure a System-Timestamped Channel Using Oracle Event Processing IDE for Eclipse 10-7	
How to Configure an Application-Timestamped Channel Using Oracle Event Processing IDE for Eclipse 10-11	
How to Create a Channel Component Configuration File Manually .....	10-14
<b>Example Channel Configuration Files.....</b>	<b>10-17</b>
Channel Component Configuration File .....	10-17
Channel EPN Assembly File.....	10-18

## 11 Integrating the Java Message Service

<b>Overview of JMS Adapter Configuration .....</b>	<b>11-1</b>
JMS Service Providers.....	11-1
Inbound JMS Adapter .....	11-2
Conversion Between JMS Messages and Event Types .....	11-2
Single and Multi-threaded Inbound JMS Adapters.....	11-3
Configuring a JMS Adapter for Durable Subscriptions .....	11-3
Outbound JMS Adapter .....	11-4
<b>Configuring a JMS Adapter for a JMS Service Provider.....</b>	<b>11-4</b>
How to Configure a JMS Adapter Using the Oracle Event Processing IDE for Eclipse.....	11-5
How to Configure a JMS Adapter Manually .....	11-5
How to Configure a JMS Adapter for Oracle WebLogic Server JMS Manually .....	11-7
How to Configure a JMS Adapter for Tibco EMS JMS Manually.....	11-10
<b>Creating a Custom Converter Between JMS Messages and Event Types.....</b>	<b>11-13</b>
How to Create a Custom Converter for the Inbound JMS Adapter .....	11-13
How to Create a Custom Converter for the Outbound JMS Adapter .....	11-14
<b>Encrypting Passwords in the JMS Adapter Component Configuration File .....</b>	<b>11-15</b>



How to Encrypt Passwords in the JMS Adapter Component Configuration File.....	11-16
<b>Configuring the JMS Adapter EPN Assembly File .....</b>	<b>11-17</b>
JMS Inbound Adapter EPN Assembly File Configuration .....	11-17
JMS Outbound Adapter EPN Assembly File Configuration.....	11-18
<b>Configuring the JMS Adapter Component Configuration File .....</b>	<b>11-19</b>
JMS Inbound Adapter Component Configuration .....	11-19
JMS Outbound Adapter Component Configuration .....	11-22

## 12 Integrating an HTTP Publish-Subscribe Server

<b>Overview of HTTP Publish-Subscribe Server Adapter Configuration.....</b>	<b>12-1</b>
Overview of the Built-In Pub-Sub Adapter for Publishing .....	12-2
Local Publishing.....	12-2
Remote Publishing.....	12-3
Overview of the Built-In Pub-Sub Adapter for Subscribing.....	12-4
Converting Between JSON Messages and Event Types.....	12-5
<b>Configuring an HTTP Pub-Sub Adapter.....</b>	<b>12-5</b>
How to Configure an HTTP Pub-Sub Adapter Using the Oracle Event Processing IDE for Eclipse 12-5	
How to Configure an HTTP Pub-Sub Adapter Manually .....	12-6
<b>Creating a Custom Converter Between the HTTP Pub-Sub Messages and Event Types .....</b>	<b>12-9</b>
<b>Configuring the HTTP Pub-Sub Adapter EPN Assembly File .....</b>	<b>12-10</b>
HTTP Pub-Sub Adapter for Publishing EPN Assembly File Configuration.....	12-10
HTTP Pub-Sub Adapter for Subscribing EPN Assembly File Configuration.....	12-12
<b>Configuring the HTTP Pub-Sub Adapter Component Configuration File.....</b>	<b>12-13</b>
HTTP Pub-Sub Adapter for Publishing Component Configuration.....	12-13
HTTP Pub-Sub Adapter for Subscribing Component Configuration .....	12-15

## 13 Integrating a Cache

<b>Overview of Integrating a Cache.....</b>	<b>13-1</b>
Caching Implementations Supported by Oracle Event Processing.....	13-2
Overview of Cache Configuration.....	13-3
Caching Use Cases .....	13-4
Caching APIs .....	13-6
<b>Configuring an Oracle Coherence Caching System and Cache .....</b>	<b>13-6</b>
Configuring the Oracle Coherence Caching System and Caches .....	13-7
The coherence-cache-config.xml File .....	13-8
The tangosol-coherence-override.xml File .....	13-10
Configuring a Shared Oracle Coherence Cache .....	13-10
<b>Configuring an Oracle Event Processing Local Caching System and Cache.....</b>	<b>13-11</b>
Configuring an Oracle Event Processing Caching System .....	13-11
<b>Configuring a Third-Party Caching System and Cache .....</b>	<b>13-14</b>
<b>Adding Caching to an Event Processing Network.....</b>	<b>13-16</b>
Adding the Caching System and Caches to an EPN.....	13-16
Configuring a Cache for Reuse Among Applications .....	13-17
Configuring a Cache as an Event Listener .....	13-17
Specifying the Key Used to Index a Cache.....	13-17

Specifying a Key Property in EPN Assembly File .....	13-18
Using a Metadata Annotation to Specify a Key .....	13-18
Specifying a Composite Key .....	13-19
Configuring a Cache as an Event Source.....	13-19
Exchanging Data Between a Cache and Another Data Source .....	13-20
Loading Cache Data from a Read-Only Data Source .....	13-20
Exchanging Data with a Read-Write Data Source .....	13-21
<b>Accessing a Cache from Application Code .....</b>	<b>13-23</b>
Accessing a Cache from an Oracle CQL Statement .....	13-23
How to Access a Cache from an Oracle CQL Statement.....	13-24
Accessing a Cache From an EPL Statement .....	13-26
How To Access a Cache from an EPL Statement .....	13-26
Accessing a Cache from an Adapter .....	13-27
Accessing a Cache From a Business POJO .....	13-28
Accessing a Cache From an Oracle CQL User-Defined Function.....	13-29
Accessing a Cache From an EPL User-Defined Function .....	13-30
Accessing a Cache Using JMX.....	13-31
How to Access a Cache With JMX Using Oracle Event Processing Visualizer .....	13-31
How to Access a Cache With JMX Using Java.....	13-31

## 14 Integrating Web Services

Understanding Oracle Event Processing and Web Services .....	14-1
How to Invoke a Web Service From an Oracle Event Processing Application .....	14-1
How to Expose an Oracle Event Processing Application as a Web Service.....	14-2

## 15 Integrating an External Component Using a Custom Adapter

Overview of Custom Adapters .....	15-1
Implementing a Custom Adapter.....	15-2
Example: Input Adapter Implementation .....	15-3
Implementing Support for Thread and Work Management .....	15-7
Improving Scalability with Multi-Threaded Adapters.....	15-7
Suspending and Resuming Adapter Event Processing.....	15-8
Passing Login Credentials from an Adapter to a Data Feed Provider .....	15-8
How to Pass Static Login Credentials to the Data Feed Provider.....	15-9
How to Pass Dynamic Login Credentials to the Data Feed Provider .....	15-10
How to Access Login Credentials From an Adapter at Runtime .....	15-11
Configuring a Custom Adapter .....	15-12
Configuring a Custom Adapter in an EPN Assembly File .....	15-13
Configuring a Custom Adapter in a Component Configuration File .....	15-13
Creating a Custom Adapter Factory.....	15-14

## 16 Handling Events with Java

Roles for Java Code in an Event Processing Network.....	16-1
Handling Events with Sources and Sinks .....	16-2
Implementing an Event Sink .....	16-4
Implementing StreamSink or BatchStreamSink .....	16-4

Implementing RelationSink or BatchRelationSink.....	16-5
Implementing an Event Source .....	16-6
Implementing StreamSource .....	16-7
Implementing RelationSource.....	16-8
<b>Configuring Java Classes as Beans .....</b>	<b>16-9</b>
Configuring a Java Class as an Event Bean .....	16-10
Configuring an Event Bean in an EPN Assembly File .....	16-10
Configuring an Event Bean in a Component Configuration File .....	16-11
Creating an Event Bean Factory.....	16-12
Configuring a Java Class as a Spring Bean.....	16-13
Supporting Spring Bean Characteristics.....	16-13
<b>17 Querying an Event Stream with Oracle CQL</b>	
<b>Overview of Oracle CQL Processor Configuration .....</b>	<b>17-1</b>
Controlling Which Queries Output to a Downstream Channel .....	17-3
<b>Configuring an Oracle CQL Processor.....</b>	<b>17-3</b>
How to Configure an Oracle CQL Processor Using Oracle Event Processing IDE for Eclipse .....	17-3
How to Create an Oracle CQL Processor Component Configuration File Manually .....	17-4
<b>Configuring an Oracle CQL Processor Table Source .....</b>	<b>17-7</b>
How to Configure an Oracle CQL Processor Table Source Using Oracle Event Processing IDE for Eclipse .....	17-8
<b>Configuring an Oracle CQL Processor Cache Source .....</b>	<b>17-11</b>
<b>Configuring an Oracle CQL Processor for Parallel Query Execution.....</b>	<b>17-11</b>
Setting Up Parallel Query Execution Support .....	17-11
Using the ordering-constraint Attribute .....	17-12
Using partition-order-capacity with Partitioning Queries.....	17-13
Limitations .....	17-14
<b>Handling Faults .....</b>	<b>17-14</b>
Implementing a Fault Handler Class .....	17-15
Registering a Fault Handler.....	17-17
<b>Example Oracle CQL Processor Configuration Files .....</b>	<b>17-17</b>
Oracle CQL Processor Component Configuration File .....	17-17
Oracle CQL Processor EPN Assembly File .....	17-18
<b>18 Configuring Applications With Data Cartridges</b>	
Understanding Data Cartridge Application Context .....	18-1
How to Configure Oracle Spatial Application Context.....	18-1
How to Configure Oracle JDBC Data Cartridge Application Context .....	18-3
<b>19 Querying an Event Stream with Oracle EPL</b>	
<b>Overview of EPL Processor Component Configuration.....</b>	<b>19-1</b>
<b>Configuring an EPL Processor .....</b>	<b>19-3</b>
How to Configure an EPL Processor Manually.....	19-3
<b>Configuring an EPL Processor Cache Source.....</b>	<b>19-6</b>
<b>Example EPL Processor Configuration Files.....</b>	<b>19-6</b>

EPL Processor Component Configuration File.....	19-6
EPL Processor EPN Assembly File .....	19-6

## 20 Configuring Event Record and Playback

<b>Overview of Configuring Event Record and Playback.....</b>	<b>20-1</b>
Storing Events in the Persistent Event Store .....	20-2
Default Persistent Event Store.....	20-2
Custom Persistent Event Store .....	20-2
Persistent Event Store Schema .....	20-2
Recording Events .....	20-3
Playing Back Events.....	20-3
Querying Stored Events .....	20-3
Record and Playback Example.....	20-4
<b>Configuring Event Record and Playback in Your Application .....</b>	<b>20-4</b>
Configuring an Event Store for Oracle Event Processing Server .....	20-4
Configuring a Component to Record Events.....	20-5
Configuring a Component to Playback Events .....	20-8
Starting and Stopping the Record and Playback of Events .....	20-10
Description of the Berkeley Database Schema.....	20-11
<b>Creating a Custom Event Store Provider .....</b>	<b>20-12</b>

## 21 Testing Applications With the Load Generator and csvgen Adapter

<b>Overview of Testing Applications With the Load Generator and csvgen Adapter .....</b>	<b>21-1</b>
<b>Configuring and Running the Load Generator Utility.....</b>	<b>21-1</b>
<b>Creating a Load Generator Property File.....</b>	<b>21-2</b>
<b>Creating a Data Feed File .....</b>	<b>21-3</b>
<b>Configuring the csvgen Adapter in Your Application.....</b>	<b>21-4</b>

## 22 Testing Applications With the Event Inspector

<b>Overview of Testing Applications With the Event Inspector .....</b>	<b>22-1</b>
Tracing Events .....	22-1
Injecting Events .....	22-2
Event Inspector Event Types .....	22-2
Event Inspector HTTP Publish-Subscribe Channel and Server .....	22-3
Event Inspector Clients .....	22-4
Oracle Event Processing Visualizer.....	22-4
<b>Configuring the Event Inspector HTTP Pub-Sub Server .....</b>	<b>22-4</b>
How to Configure a Local Event Inspector HTTP Pub-Sub Server .....	22-5
How to Configure a Remote Event Inspector HTTP Pub-Sub Server .....	22-5
<b>Injecting Events .....</b>	<b>22-6</b>
<b>Tracing Events.....</b>	<b>22-7</b>

## Part IV Completing and Refining Oracle Event Processing Applications

### 23 Assembling and Deploying Oracle Event Processing Applications

<b>Overview of Application Assembly and Deployment .....</b>	<b>23-1</b>
--	-------------

Applications .....	23-2
Application Dependencies.....	23-2
Private Application Dependencies .....	23-2
Shared Application Dependencies .....	23-3
Native Code Dependencies .....	23-3
Application Libraries .....	23-3
Library Directory .....	23-4
Library Extensions Directory .....	23-4
Creating Application Libraries .....	23-5
Deployment and Deployment Order .....	23-5
Configuration History Management.....	23-6
<b>Assembling an Oracle Event Processing Application.....</b>	<b>23-6</b>
Assembling an Oracle Event Processing Application Using Oracle Event Processing IDE for Eclipse 23-7	
Assembling an Oracle Event Processing Application Manually .....	23-7
Creating the MANIFEST.MF File .....	23-8
Accessing Third-Party JAR Files.....	23-10
Accessing Third-Party JAR Files Using Bundle-Classpath .....	23-10
Accessing Third-Party JAR Files Using -Xbootclasspath.....	23-10
Assembling Applications With Foreign Stages .....	23-11
Assembling a Custom Adapter or Event Bean in Its Own Bundle.....	23-12
How to Assemble a Custom Adapter in its Own Bundle .....	23-12
How to Assemble an Event Bean in its Own Bundle .....	23-13
<b>Managing Application Libraries .....</b>	<b>23-14</b>
How to Define the Application Library Directory Using Oracle Event Processing IDE for Eclipse 23-14	
How to Configure an Absolute Path.....	23-15
How to Extend a Path Variable.....	23-16
How to Create an Application Library Using bundler.sh.....	23-18
How to Create an Application Library Using Oracle Event Processing IDE for Eclipse ...	23-21
How to Update an Application Library Using Oracle Event Processing IDE for Eclipse..	23-30
How to View an Application Library Using the Oracle Event Processing Visualizer.....	23-31
<b>Managing Log Message Catalogs .....</b>	<b>23-32</b>
Using Message Catalogs With Oracle Event Processing Server.....	23-32
Message Catalog Hierarchy.....	23-33
Guidelines for Naming Message Catalogs.....	23-33
Using Message Arguments.....	23-34
Message Catalog Formats .....	23-35
Log Message Catalog .....	23-35
Simple Text Catalog.....	23-35
Locale-Specific Catalog .....	23-36
Message Catalog Localization.....	23-36
How to Parse a Message Catalog to Generate Logger and TextFormatter Classes for Localization 23-37	
<b>Deploying Oracle Event Processing Applications.....</b>	<b>23-38</b>
How to Deploy an Oracle Event Processing Application Using Oracle Event Processing IDE for Eclipse 23-39	

How to Deploy an Oracle Event Processing Application Using Oracle Event Processing Visualizer	23-39
How to Deploy an Oracle Event Processing Application Using the Deployer Utility .....	23-40

## 24 Developing Applications for High Availability

<b>Understanding High Availability</b> .....	24-1
High Availability Architecture.....	24-1
High Availability Lifecycle and Failover.....	24-2
Secondary Failure .....	24-3
Primary Failure and Failover .....	24-3
Rejoining the High Availability Multi-Server Domain .....	24-3
Deployment Group and Notification Group .....	24-4
High Availability Components.....	24-4
High Availability Input Adapter.....	24-6
Buffering Output Adapter.....	24-6
Broadcast Output Adapter .....	24-6
Correlating Output Adapter .....	24-7
ActiveActiveGroupBean.....	24-7
High Availability and Scalability .....	24-7
High Availability and Oracle Coherence .....	24-8
Choosing a Quality of Service .....	24-9
Simple Failover.....	24-9
Simple Failover with Buffering.....	24-10
Light-Weight Queue Trimming.....	24-10
Precise Recovery with JMS .....	24-11
Designing an Oracle Event Processing Application for High Availability .....	24-12
Primary Oracle Event Processing High Availability Use Case .....	24-12
High Availability Design Patterns .....	24-13
Select the Minimum High Availability Your Application can Tolerate .....	24-13
Use Oracle Event Processing High Availability Components at All Ingress and Egress Points	24-13
Only Preserve What You Need.....	24-13
Limit Oracle Event Processing Application State .....	24-14
Choose an Adequate warm-up-window-length Time .....	24-14
Type 1 Applications.....	24-14
Type 2 Applications.....	24-15
Ensure Applications are Idempotent .....	24-15
Source Event Identity Externally.....	24-15
Understand the Importance of Event Ordering .....	24-16
Prefer Deterministic Behavior.....	24-16
Avoid Multithreading .....	24-16
Prefer Monotonic Event Identifiers.....	24-17
Write Oracle CQL Queries with High Availability in Mind .....	24-17
Avoid Coupling Servers .....	24-17
Plan for Server Recovery .....	24-17
Oracle CQL Query Restrictions .....	24-17
Range-Based Windows .....	24-18

Tuple-Based Windows .....	24-18
Partitioned Windows .....	24-18
Sliding Windows.....	24-18
DURATION Clause and Non-Event Detection.....	24-18
Prefer Application Time .....	24-18
<b>Configuring High Availability</b> .....	24-19
Configuring High Availability Quality of Service .....	24-19
How to Configure Simple Failover .....	24-20
How to Configure Simple Failover With Buffering.....	24-23
How to Configure Light-Weight Queue Trimming.....	24-27
How to Configure Precise Recovery With JMS .....	24-35
Configuring High Availability Adapters .....	24-43
How to Configure the High Availability Input Adapter .....	24-44
High Availability Input Adapter EPN Assembly File Configuration .....	24-44
High Availability Input Adapter Component Configuration File Configuration .....	24-45
How to Configure the Buffering Output Adapter .....	24-46
Buffering Output Adapter EPN Assembly File Configuration.....	24-46
Buffering Output Adapter Component Configuration File Configuration .....	24-47
How to Configure the Broadcast Output Adapter .....	24-47
Broadcast Output Adapter EPN Assembly File Configuration.....	24-47
Broadcast Output Adapter Component Configuration File Configuration.....	24-48
How to Configure the Correlating Output Adapter.....	24-49
Correlating Output Adapter EPN Assembly File Configuration .....	24-49
Correlating Output Adapter Component Configuration File Configuration.....	24-50

## 25 Developing Scalable Applications

<b>Understanding Scalability</b> .....	25-1
Scalability Options .....	25-1
Scalability and High Availability .....	25-1
Scalability Components.....	25-2
EventPartitioner .....	25-2
EventPartitioner Implementation.....	25-2
EventPartitioner Initialization .....	25-3
EventPartitioner Threading.....	25-3
EventPartitioner Restrictions .....	25-3
ActiveActiveGroupBean .....	25-3
Scalability in an Oracle Event Processing Application Using the ActiveActiveGroupBean Without High Availability 25-4	
Scalability in an Oracle Event Processing Application Using the ActiveActiveGroupBean With High Availability 25-6	
<b>Configuring Scalability</b> .....	25-7
Configuring Scalability With a Channel EventPartitioner.....	25-7
How to Configure Scalability With the Default Channel EventPartitioner .....	25-8
How to Configure Scalability With a Custom Channel EventPartitioner .....	25-10
Configuring Scalability With the ActiveActiveGroupBean.....	25-14
How to Configure Scalability in a JMS Application Without Oracle Event Processing High Availability 25-15	

How to Configure Scalability in a JMS Application With Oracle Event Processing High Availability	25-17
How to Configure the ActiveActiveGroupBean Group Pattern Match.....	25-23

## 26 Extending Component Configuration

<b>Overview of Extending Component Configuration</b> .....	26-1
Extending Component Configuration Using Annotations.....	26-2
Extending Component Configuration Using an XSD.....	26-2
<b>Extending Component Configuration</b> .....	26-2
How to Extend Component Configuration Using Annotations.....	26-2
How to Extend Component Configuration Using an XSD.....	26-4
Creating the XSD Schema File.....	26-6
Complete Example of an Extended XSD Schema File.....	26-8
<b>Programming Access to the Configuration of a Custom Adapter or Event Bean</b> .....	26-9
How to Access Component Configuration Using Resource Injection.....	26-9
How to Access Component Configuration Using Lifecycle Callbacks.....	26-10
Lifecycle Callback Annotations.....	26-11
Lifecycle.....	26-11

## 27 Performance Tuning

<b>EPN Performance Tuning</b> .....	27-1
Event Partitioner Channel.....	27-1
Batching Channel.....	27-1
Scalability Using the ActiveActiveGroupBean.....	27-1
<b>High Availability Performance Tuning</b> .....	27-2
Host Configuration.....	27-2
High Availability Input Adapter and Quality of Service.....	27-2
High Availability Input Adapter Configuration.....	27-2
Broadcast Output Adapter Configuration.....	27-2
Oracle Coherence Performance Tuning Options.....	27-3
Oracle Coherence Heartbeat Frequency.....	27-3
Oracle Coherence Serialization.....	27-3

## Part V Appendices

### A Additional Information about Spring and OSGi

### B Oracle Event Processing Schemas

<b>EPN Assembly Schema spring-wlevs-v11_1_1_6.xsd</b> .....	B-1
Example EPN Assembly File.....	B-1
<b>Component Configuration Schema wlevs_application_config.xsd</b> .....	B-2
Example Component Configuration File.....	B-2
<b>Deployment Schema deployment.xsd</b> .....	B-3
Example Deployment XML File.....	B-3
<b>Server Configuration Schema wlevs_server_config.xsd</b> .....	B-3
Example Server Configuration XML File.....	B-4



## C Schema Reference: EPN Assembly spring-wlevs-v11\_1\_1\_6.xsd

Overview of the Oracle Event Processing Application Assembly Elements .....	C-2
Element Hierarchy .....	C-2
Example of an EPN Assembly File That Uses Oracle Event Processing Elements .....	C-3
<b>wlevs:adapter</b> .....	C-3
Child Elements .....	C-3
Attributes.....	C-3
Example .....	C-5
<b>wlevs:application-timestamped</b> .....	C-5
Child Elements .....	C-5
Attributes.....	C-5
Example .....	C-6
<b>wlevs:cache</b> .....	C-6
Child Elements .....	C-6
Attributes.....	C-7
Example .....	C-7
<b>wlevs:cache-listener</b> .....	C-8
Attributes.....	C-8
Example .....	C-8
<b>wlevs:cache-loader</b> .....	C-8
Attributes.....	C-8
Example .....	C-9
<b>wlevs:cache-source</b> .....	C-9
Attributes.....	C-9
Example .....	C-9
<b>wlevs:cache-store</b> .....	C-10
Attributes.....	C-10
Example .....	C-10
<b>wlevs:caching-system</b> .....	C-10
Child Elements .....	C-10
Attributes.....	C-10
Example .....	C-11
<b>wlevs:channel</b> .....	C-11
Child Elements .....	C-12
Attributes.....	C-12
Example .....	C-13
<b>wlevs:class</b> .....	C-13
Example .....	C-13
<b>wlevs:event-bean</b> .....	C-14
Child Elements .....	C-14
Attributes.....	C-14
Example .....	C-15
<b>wlevs:event-type</b> .....	C-16
Child Elements .....	C-16
Attributes.....	C-16
Example .....	C-16
<b>wlevs:event-type-repository</b> .....	C-17

Child Elements .....	C-17
Example .....	C-17
<b>wlevs:expression</b> .....	C-17
Example .....	C-18
<b>wlevs:factory</b> .....	C-18
Attributes .....	C-18
Example .....	C-18
<b>wlevs:function</b> .....	C-18
Attributes .....	C-19
Example .....	C-20
Single-Row User-Defined Function on an Oracle CQL Processor .....	C-20
Single-Row User-Defined Function on an EPL Processor .....	C-21
Aggregate User-Defined Function on an Oracle CQL Processor .....	C-21
Aggregate User-Defined Function on an EPL Processor .....	C-23
Specifying the Implementation Class: Nested Bean or Reference .....	C-24
<b>wlevs:instance-property</b> .....	C-25
Child Elements .....	C-25
Attributes .....	C-26
Example .....	C-26
<b>wlevs:listener</b> .....	C-26
Attributes .....	C-27
Example .....	C-27
<b>wlevs:metadata</b> .....	C-27
Child Elements .....	C-27
Attributes .....	C-27
Example .....	C-28
<b>wlevs:processor</b> .....	C-28
Child Elements .....	C-28
Attributes .....	C-28
Example .....	C-29
<b>wlevs:properties</b> .....	C-29
Child Elements .....	C-29
Attributes .....	C-29
Example .....	C-30
<b>wlevs:property</b> .....	C-30
Attributes .....	C-30
Example .....	C-31
<b>wlevs:property</b> .....	C-31
Child Elements .....	C-31
Attributes .....	C-32
Example .....	C-32
<b>wlevs:source</b> .....	C-32
Attributes .....	C-32
Example .....	C-33
<b>wlevs:table</b> .....	C-33
Attributes .....	C-33
Example .....	C-33

<b>wlevs:table-source</b> .....	C-34
Attributes.....	C-34
Example .....	C-34

## **D Schema Reference: Component Configuration wlevs\_application\_config.xsd**

<b>Overview of the Oracle Event Processing Component Configuration Elements</b> .....	D-4
Element Hierarchy .....	D-4
Example of an Oracle Event Processing Component Configuration File .....	D-13
<b>accept-backlog</b> .....	D-14
Child Elements .....	D-14
Attributes.....	D-14
Example .....	D-14
<b>active</b> .....	D-14
Child Elements .....	D-15
Attributes.....	D-15
Example .....	D-15
<b>adapter</b> .....	D-15
Child Elements .....	D-15
Attributes.....	D-15
Example .....	D-15
<b>amount</b> .....	D-16
Child Elements .....	D-16
Attributes.....	D-16
Example .....	D-16
<b>application</b> .....	D-17
Child Elements .....	D-17
Attributes.....	D-17
Example .....	D-17
<b>average-interval</b> .....	D-17
Child Elements .....	D-18
Attributes.....	D-18
Example .....	D-18
<b>average-latency</b> .....	D-18
Child Elements .....	D-18
Attributes.....	D-19
Example .....	D-19
<b>batch-size</b> .....	D-19
Child Elements .....	D-19
Attributes.....	D-19
Example .....	D-19
<b>batch-time-out</b> .....	D-20
Child Elements .....	D-20
Attributes.....	D-20
Example .....	D-20
<b>binding</b> .....	D-20
Child Elements .....	D-20
Attributes.....	D-20

Example .....	D-21
<b>bindings (jms-adapter)</b> .....	D-21
Child Elements .....	D-21
Attributes.....	D-21
Example .....	D-22
<b>bindings (processor)</b> .....	D-22
Child Elements .....	D-22
Attributes.....	D-22
Example .....	D-23
<b>buffer-size</b> .....	D-23
Child Elements .....	D-23
Attributes.....	D-23
Example .....	D-23
<b>buffer-write-attempts</b> .....	D-24
Child Elements .....	D-24
Attributes.....	D-24
Example .....	D-24
<b>buffer-write-timeout</b> .....	D-24
Child Elements .....	D-25
Attributes.....	D-25
Example .....	D-25
<b>cache</b> .....	D-25
Child Elements .....	D-25
Attributes.....	D-26
Example .....	D-26
<b>caching-system</b> .....	D-26
Child Elements .....	D-26
Attributes.....	D-26
Example .....	D-27
<b>channel</b> .....	D-27
Child Elements .....	D-27
Attributes.....	D-27
Example .....	D-27
<b>channel (http-pub-sub-adapter Child Element)</b> .....	D-28
Child Elements .....	D-28
Attributes.....	D-28
Example .....	D-28
<b>channel-name</b> .....	D-28
Child Elements .....	D-28
Attributes.....	D-29
Example .....	D-29
<b>coherence-cache-config</b> .....	D-29
Child Elements .....	D-29
Attributes.....	D-29
Example .....	D-29
<b>coherence-caching-system</b> .....	D-29
Child Elements .....	D-30

Attributes.....	D-30
Example .....	D-30
<b>coherence-cluster-config</b> .....	D-30
Child Elements .....	D-30
Attributes.....	D-30
Example .....	D-30
<b>collect-interval</b> .....	D-31
Child Elements .....	D-31
Attributes.....	D-31
Example .....	D-31
<b>concurrent-consumers</b> .....	D-31
Child Elements .....	D-32
Attributes.....	D-32
Example .....	D-32
<b>connection-jndi-name</b> .....	D-32
Child Elements .....	D-32
Attributes.....	D-32
Example .....	D-32
<b>connection-encrypted-password</b> .....	D-33
Child Elements .....	D-33
Attributes.....	D-33
Example .....	D-33
<b>connection-password</b> .....	D-33
Child Elements .....	D-34
Attributes.....	D-34
Example .....	D-34
<b>connection-user</b> .....	D-34
Child Elements .....	D-34
Attributes.....	D-34
Example .....	D-34
<b>database</b> .....	D-35
Child Elements .....	D-35
Attributes.....	D-35
Example .....	D-35
<b>dataset-name</b> .....	D-35
Child Elements .....	D-35
Attributes.....	D-36
Example .....	D-36
<b>delivery-mode</b> .....	D-36
Child Elements .....	D-36
Attributes.....	D-36
Example .....	D-36
<b>destination-jndi-name</b> .....	D-36
Child Elements .....	D-36
Attributes.....	D-37
Example .....	D-37
<b>destination-name</b> .....	D-37

Child Elements .....	D-37
Attributes.....	D-37
Example .....	D-37
<b>destination-type</b> .....	D-37
Child Elements .....	D-37
Attributes.....	D-38
Example .....	D-38
<b>diagnostic-profiles</b> .....	D-38
Child Elements .....	D-38
Attributes.....	D-38
Example .....	D-38
<b>direction</b> .....	D-39
Child Elements .....	D-39
Attributes.....	D-39
Example .....	D-39
<b>durable-subscription</b> .....	D-39
Child Elements .....	D-40
Attributes.....	D-40
Example .....	D-40
<b>durable-subscription-name</b> .....	D-40
Child Elements .....	D-40
Attributes.....	D-40
Example .....	D-40
<b>duration</b> .....	D-41
Child Elements .....	D-41
Attributes.....	D-41
Example .....	D-41
<b>enabled</b> .....	D-41
Child Elements .....	D-41
Attributes.....	D-42
Example .....	D-42
<b>encrypted-password</b> .....	D-42
Child Elements .....	D-42
Attributes.....	D-42
Example .....	D-43
<b>end</b> .....	D-43
Child Elements .....	D-43
Attributes.....	D-43
Example .....	D-43
<b>end-location</b> .....	D-44
Child Elements .....	D-44
Attributes.....	D-44
Example .....	D-44
<b>event-bean</b> .....	D-44
Child Elements .....	D-45
Attributes.....	D-45
Example .....	D-45

<b>event-type</b> .....	D-45
Child Elements .....	D-46
Attributes.....	D-46
Example .....	D-46
<b>event-type-list</b> .....	D-46
Child Elements .....	D-46
Attributes.....	D-46
Example .....	D-46
<b>eviction-policy</b> .....	D-46
Child Elements .....	D-47
Attributes.....	D-47
Example .....	D-47
<b>fail-when-rejected</b> .....	D-47
Child Elements .....	D-47
Attributes.....	D-47
Example .....	D-47
<b>group-binding</b> .....	D-48
Child Elements .....	D-48
Attributes.....	D-48
Example .....	D-48
<b>heartbeat</b> .....	D-49
Child Elements .....	D-49
Attributes.....	D-49
Example .....	D-49
<b>http-pub-sub-adapter</b> .....	D-49
Child Elements .....	D-49
Attributes.....	D-50
Example .....	D-50
<b>idle-time</b> .....	D-50
Child Elements .....	D-50
Attributes.....	D-50
Example .....	D-51
<b>inject-parameters</b> .....	D-51
Child Elements .....	D-51
Attributes.....	D-51
Example .....	D-51
<b>json-adapter</b> .....	D-52
Child Elements .....	D-52
Attributes.....	D-52
Example .....	D-52
<b>jndi-factory</b> .....	D-53
Child Elements .....	D-53
Attributes.....	D-53
Example .....	D-53
<b>jndi-provider-url</b> .....	D-53
Child Elements .....	D-53
Attributes.....	D-53

Example .....	D-53
<b>listeners</b> .....	D-54
Child Elements .....	D-54
Attributes.....	D-54
Example .....	D-54
<b>location</b> .....	D-55
Child Elements .....	D-55
Attributes.....	D-55
Example .....	D-55
<b>max-latency</b> .....	D-55
Child Elements .....	D-56
Attributes.....	D-56
Example .....	D-56
<b>max-size</b> .....	D-56
Child Elements .....	D-56
Attributes.....	D-56
Example .....	D-57
<b>max-threads</b> .....	D-57
Child Elements .....	D-57
Attributes.....	D-57
Example .....	D-57
<b>message-selector</b> .....	D-57
Child Elements .....	D-58
Attributes.....	D-58
Example .....	D-58
<b>name</b> .....	D-58
Child Elements .....	D-58
Attributes.....	D-58
Example .....	D-58
<b>netio</b> .....	D-59
Child Elements .....	D-59
Attributes.....	D-59
Example .....	D-59
<b>num-threads</b> .....	D-59
Child Elements .....	D-59
Attributes.....	D-59
Example .....	D-59
<b>offer-timeout</b> .....	D-59
Child Elements .....	D-60
Attributes.....	D-60
Example .....	D-60
<b>param</b> .....	D-60
Child Elements .....	D-60
Attributes.....	D-60
Example .....	D-60
<b>parameter</b> .....	D-61
Child Elements .....	D-61



Attributes.....	D-61
Example .....	D-61
<b>params</b> .....	D-62
Child Elements .....	D-62
Attributes.....	D-62
Example .....	D-62
<b>partition-order-capacity</b> .....	D-63
Child Elements .....	D-63
Attributes.....	D-63
Example .....	D-63
<b>password</b> .....	D-63
Child Elements .....	D-63
Attributes.....	D-64
Example .....	D-64
<b>playback-parameters</b> .....	D-64
Child Elements .....	D-64
Attributes.....	D-64
Example .....	D-64
<b>playback-speed</b> .....	D-65
Child Elements .....	D-65
Attributes.....	D-65
Example .....	D-65
<b>processor (EPL)</b> .....	D-65
Child Elements .....	D-65
Attributes.....	D-66
Example .....	D-66
<b>processor (Oracle CQL)</b> .....	D-66
Child Elements .....	D-66
Attributes.....	D-67
Example .....	D-67
<b>profile</b> .....	D-68
Child Elements .....	D-68
Attributes.....	D-68
Example .....	D-68
<b>provider-name</b> .....	D-69
Child Elements .....	D-69
Attributes.....	D-69
Example .....	D-69
<b>query</b> .....	D-69
Child Elements .....	D-70
Attributes.....	D-70
Example .....	D-70
<b>record-parameters</b> .....	D-71
Child Elements .....	D-71
Attributes.....	D-71
Example .....	D-71
<b>repeat</b> .....	D-72

Child Elements .....	D-72
Attributes.....	D-72
Example .....	D-72
<b>rule</b> .....	D-72
Child Elements .....	D-72
Attributes.....	D-72
Example .....	D-73
<b>rules</b> .....	D-73
Child Elements .....	D-73
Attributes.....	D-74
Example .....	D-74
<b>schedule-time-range</b> .....	D-74
Child Elements .....	D-74
Attributes.....	D-74
Example .....	D-75
<b>schedule-time-range-offset</b> .....	D-75
Child Elements .....	D-75
Attributes.....	D-75
Example .....	D-75
<b>selector</b> .....	D-76
Child Elements .....	D-77
Attributes.....	D-77
Example .....	D-77
<b>server-context-path</b> .....	D-77
Child Elements .....	D-78
Attributes.....	D-78
Example .....	D-78
<b>server-url</b> .....	D-78
Child Elements .....	D-78
Attributes.....	D-78
Example .....	D-78
<b>session-ack-mode-name</b> .....	D-79
Child Elements .....	D-79
Attributes.....	D-79
Example .....	D-79
<b>session-transacted</b> .....	D-79
Child Elements .....	D-79
Attributes.....	D-80
Example .....	D-80
<b>stage</b> .....	D-80
Child Elements .....	D-80
Attributes.....	D-80
Example .....	D-80
<b>start</b> .....	D-81
Child Elements .....	D-81
Attributes.....	D-81
Example .....	D-81

<b>start-location</b> .....	D-81
Child Elements .....	D-82
Attributes.....	D-82
Example .....	D-82
<b>start-stage</b> .....	D-82
Child Elements .....	D-83
Attributes.....	D-83
Example .....	D-83
<b>store-policy-parameters</b> .....	D-83
Child Elements .....	D-83
Attributes.....	D-83
Example .....	D-83
<b>stream</b> .....	D-84
Child Elements .....	D-84
Attributes.....	D-84
Example .....	D-84
<b>symbol</b> .....	D-84
Child Elements .....	D-84
Attributes.....	D-85
Example .....	D-85
<b>symbols</b> .....	D-85
Child Elements .....	D-85
Attributes.....	D-85
Example .....	D-85
<b>threshold</b> .....	D-85
Child Elements .....	D-86
Attributes.....	D-86
Example .....	D-86
<b>throughput</b> .....	D-86
Child Elements .....	D-86
Attributes.....	D-87
Example .....	D-87
<b>throughput-interval</b> .....	D-87
Child Elements .....	D-87
Attributes.....	D-87
Example .....	D-87
<b>time-range</b> .....	D-88
Child Elements .....	D-88
Attributes.....	D-88
Example .....	D-88
<b>time-range-offset</b> .....	D-89
Child Elements .....	D-89
Attributes.....	D-89
Example .....	D-89
<b>time-to-live</b> .....	D-89
Child Elements .....	D-90
Attributes.....	D-90

Example .....	D-90
<b>trace-parameters</b> .....	D-90
Child Elements .....	D-90
Attributes.....	D-90
Example .....	D-90
<b>unit</b> .....	D-91
Child Elements .....	D-91
Attributes.....	D-91
Example .....	D-91
<b>user</b> .....	D-92
Child Elements .....	D-92
Attributes.....	D-92
Example .....	D-92
<b>value</b> .....	D-92
Child Elements .....	D-92
Attributes.....	D-92
Example .....	D-92
<b>view</b> .....	D-93
Child Elements .....	D-93
Attributes.....	D-93
Example .....	D-94
<b>work-manager</b> .....	D-94
Child Elements .....	D-94
Attributes.....	D-94
Example .....	D-94
<b>work-manager-name</b> .....	D-95
Child Elements .....	D-95
Attributes.....	D-95
Example .....	D-95
<b>write-behind</b> .....	D-95
Child Elements .....	D-95
Attributes.....	D-96
Example .....	D-96
<b>write-none</b> .....	D-96
Child Elements .....	D-96
Attributes.....	D-96
Example .....	D-96
<b>write-through</b> .....	D-97
Child Elements .....	D-97
Attributes.....	D-97
Example .....	D-97

## **E Schema Reference: Deployment deployment.xsd**

<b>Overview of the Oracle Event Processing Deployment Elements</b> .....	E-1
Element Hierarchy .....	E-1
Example of an Oracle Event Processing Deployment Configuration File.....	E-1
<b>wlevs:deployment</b> .....	E-2

Child Elements .....	E-2
Attributes.....	E-2
Example .....	E-2

## **F Schema Reference: Server Configuration wlevs\_server\_config.xsd**

<b>Overview of the Oracle Event Processing Server Configuration Elements.....</b>	<b>F-2</b>
Element Hierarchy .....	F-2
Example of an Oracle Event Processing Server Configuration File.....	F-3
<b>auth-constraint .....</b>	<b>F-5</b>
Child Elements .....	F-5
Attributes.....	F-6
Example .....	F-6
<b>bdb-config.....</b>	<b>F-6</b>
Child Elements .....	F-6
Attributes.....	F-7
Example .....	F-7
<b>channels.....</b>	<b>F-7</b>
Child Elements .....	F-7
Attributes.....	F-7
Example .....	F-7
<b>channel-constraints .....</b>	<b>F-8</b>
Child Elements .....	F-8
Attributes.....	F-8
Example .....	F-8
<b>channel-resource-collection.....</b>	<b>F-9</b>
Child Elements .....	F-9
Attributes.....	F-9
Example .....	F-9
<b>cluster.....</b>	<b>F-10</b>
Child Elements .....	F-10
Attributes.....	F-11
Example .....	F-12
<b>connection-pool-params.....</b>	<b>F-12</b>
Child Elements .....	F-12
Attributes.....	F-14
Example .....	F-14
<b>cql .....</b>	<b>F-15</b>
Child Elements .....	F-15
Attributes.....	F-15
Example .....	F-15
<b>data-source .....</b>	<b>F-15</b>
Child Elements .....	F-15
Attributes.....	F-15
Example .....	F-16
<b>data-source-params .....</b>	<b>F-16</b>
Child Elements .....	F-16
Attributes.....	F-17

Example .....	F-17
<b>driver-params</b> .....	F-18
Child Elements .....	F-18
Attributes.....	F-18
Example .....	F-19
<b>domain</b> .....	F-19
Child Elements .....	F-19
Attributes.....	F-19
Example .....	F-19
<b>debug</b> .....	F-20
Child Elements .....	F-20
Attributes.....	F-20
Example .....	F-20
<b>event-store</b> .....	F-20
Child Elements .....	F-20
Attributes.....	F-21
Example .....	F-21
<b>exported-jndi-context</b> .....	F-21
Child Elements .....	F-21
Attributes.....	F-21
Example .....	F-22
<b>http-pubsub</b> .....	F-22
Child Elements .....	F-22
Attributes.....	F-22
Example .....	F-22
<b>jetty</b> .....	F-23
Child Elements .....	F-23
Attributes.....	F-23
Example .....	F-23
<b>jetty-web-app</b> .....	F-24
Child Elements .....	F-24
Attributes.....	F-24
Example .....	F-24
<b>jmx</b> .....	F-24
Child Elements .....	F-25
Attributes.....	F-25
Example .....	F-25
<b>jndi-context</b> .....	F-25
Child Elements .....	F-25
Attributes.....	F-25
Example .....	F-26
<b>log-file</b> .....	F-26
Child Elements .....	F-26
Attributes.....	F-27
Example .....	F-27
<b>log-stdout</b> .....	F-27
Child Elements .....	F-27

Attributes.....	F-28
Example .....	F-28
<b>logging-service</b> .....	F-28
Child Elements .....	F-28
Attributes.....	F-29
Example .....	F-29
<b>message-filters</b> .....	F-29
Child Elements .....	F-29
Attributes.....	F-30
Example .....	F-30
<b>name</b> .....	F-30
Child Elements .....	F-30
Attributes.....	F-30
Example .....	F-30
<b>netio</b> .....	F-30
Child Elements .....	F-31
Attributes.....	F-31
Example .....	F-31
<b>netio-client</b> .....	F-31
Child Elements .....	F-31
Attributes.....	F-32
Example .....	F-32
<b>partition-order-capacity</b> .....	F-32
Child Elements .....	F-32
Attributes.....	F-32
Example .....	F-32
<b>path</b> .....	F-33
Child Elements .....	F-33
Attributes.....	F-33
Example .....	F-33
<b>pubsub-bean</b> .....	F-33
Child Elements .....	F-33
Attributes.....	F-34
Example .....	F-34
<b>rdbms-event-store-provider</b> .....	F-34
Child Elements .....	F-34
Attributes.....	F-35
Example .....	F-35
<b>rmi</b> .....	F-35
Child Elements .....	F-35
Attributes.....	F-36
Example .....	F-36
<b>scheduler</b> .....	F-36
Child Elements .....	F-36
Attributes.....	F-37
Example .....	F-37
<b>server-config</b> .....	F-37

Child Elements .....	F-37
Attributes.....	F-38
Example .....	F-38
<b>services</b> .....	F-39
Child Elements .....	F-39
Attributes.....	F-39
Example .....	F-39
<b>show-detail-error-message</b> .....	F-40
Child Elements .....	F-40
Attributes.....	F-40
Example .....	F-40
<b>ssl</b> .....	F-41
Child Elements .....	F-41
Attributes.....	F-42
Example .....	F-42
<b>timeout-seconds</b> .....	F-42
Child Elements .....	F-42
Attributes.....	F-42
Example .....	F-42
<b>transaction-manager</b> .....	F-43
Child Elements .....	F-43
Attributes.....	F-45
Example .....	F-45
<b>use-secure-connections</b> .....	F-45
Child Elements .....	F-46
Attributes.....	F-46
Example .....	F-46
<b>weblogic-instances</b> .....	F-46
Child Elements .....	F-46
Attributes.....	F-46
Example .....	F-47
<b>weblogic-jta-gateway</b> .....	F-47
Child Elements .....	F-47
Attributes.....	F-47
Example .....	F-47
<b>weblogic-rmi-client</b> .....	F-48
Child Elements .....	F-48
Attributes.....	F-48
Example .....	F-48
<b>work-manager</b> .....	F-48
Child Elements .....	F-48
Attributes.....	F-49
Example .....	F-49
<b>xa-params</b> .....	F-49
Child Elements .....	F-49
Attributes.....	F-51
Example .....	F-51



## G Schema Reference: Message Catalog msgcat.dtd

<b>Overview of the Message Catalog Elements</b> .....	G-1
Element Hierarchy .....	G-1
Examples .....	G-2
<b>message_catalog</b> .....	G-3
Child Elements .....	G-3
Attributes.....	G-3
Example .....	G-4
<b>logmessage</b> .....	G-5
Child Elements .....	G-5
Attributes.....	G-5
Example .....	G-7
<b>message</b> .....	G-7
Child Elements .....	G-7
Attributes.....	G-7
Example .....	G-8
<b>messagebody</b> .....	G-9
Child Elements .....	G-9
Attributes.....	G-9
Example .....	G-9
<b>messagedetail</b> .....	G-10
Child Elements .....	G-10
Attributes.....	G-10
Example .....	G-10
<b>cause</b> .....	G-10
Child Elements .....	G-11
Attributes.....	G-11
Example .....	G-11
<b>action</b> .....	G-11
Child Elements .....	G-11
Attributes.....	G-12
Example .....	G-12

## H Schema Reference: Locale Message Catalog l10n\_msgcat.dtd

<b>Overview of the Locale Message Catalog Elements</b> .....	H-1
Element Hierarchy .....	H-1
Examples .....	H-2
<b>locale_message_catalog</b> .....	H-3
Child Elements .....	H-3
Attributes.....	H-3
Example .....	H-3
<b>logmessage</b> .....	H-4
Child Elements .....	H-4
Attributes.....	H-4
Example .....	H-4
<b>message</b> .....	H-5

Child Elements .....	H-5
Attributes.....	H-5
Example .....	H-5
<b>messagebody</b> .....	H-6
Child Elements .....	H-6
Attributes.....	H-6
Example .....	H-6
<b>messagedetail</b> .....	H-6
Child Elements .....	H-6
Attributes.....	H-6
Example .....	H-7
<b>cause</b> .....	H-7
Child Elements .....	H-7
Attributes.....	H-7
Example .....	H-7
<b>action</b> .....	H-8
Child Elements .....	H-8
Attributes.....	H-8
Example .....	H-8

## I Oracle Event Processing Metadata Annotation Reference

<b>Overview of Oracle Event Processing Metadata Annotations</b> .....	I-1
Adapter Lifecycle Annotations .....	I-1
OSGi Service Reference Annotations .....	I-2
Resource Access Annotations.....	I-2
<b>com.bea.wlevs.configuration.Activate</b> .....	I-2
Example .....	I-2
<b>com.bea.wlevs.configuration.Prepare</b> .....	I-4
Example .....	I-4
<b>com.bea.wlevs.configuration.Rollback</b> .....	I-5
Example .....	I-5
<b>com.bea.wlevs.util.Service</b> .....	I-6
Attributes.....	I-6
Example .....	I-7



## List of Tables

1-1	Resource Name Resolution.....	1-17
2-1	Valid Order Workflow .....	2-12
2-2	Invalid Order Workflow .....	2-12
2-3	MATCH_RECOGNIZE Pattern Quantifiers .....	2-21
2-4	Condition Definitions.....	2-22
4-1	New Update Site Dialog Attributes .....	4-4
4-2	Oracle Event Processing IDE for Eclipse Plug-Ins .....	4-7
4-3	Oracle Event Processing IDE for Eclipse Plug-Ins .....	4-11
5-1	Oracle Event Processing Project Artifacts .....	5-2
5-2	Create an Oracle Event Processing Application Dialog.....	5-4
5-3	Oracle Event Processing Application Content Dialog.....	5-5
5-4	New OEP Assembly File Dialog .....	5-8
5-5	New OEP Configuration File Dialog.....	5-10
5-6	Oracle Event Processing Application Content Dialog.....	5-12
5-7	Oracle Event Processing Problem Severities.....	5-41
6-1	Eclipse and Oracle Event Processing Server Concepts.....	6-1
6-2	New Server: Define New Server Dialog (No Installed Runtimes) Attributes .....	6-5
6-3	New Server: New Oracle Event Processing v11 Runtime Dialog Attributes.....	6-6
6-4	New Server: Define New Server (Installed Runtimes) Dialog Attributes .....	6-7
6-5	New Server: New Oracle Event Processing v11 Server Dialog Attributes for a Local Server 6-9	
6-6	New Server: Define New Server Dialog (No Installed Runtimes) Attributes .....	6-12
6-7	New Server: New Oracle Event Processing v11 Runtime Dialog Attributes.....	6-13
6-8	New Server: Define New Server (Installed Runtimes) Dialog Attributes .....	6-14
6-9	New Server: Oracle Event Processing v11 Server Dialog Attributes for a Local Server .....	6-16
6-10	New Server Runtime Dialog Attributes .....	6-18
6-11	New Server Runtime Dialog Attributes .....	6-19
6-12	Add and Remove Dialog Attributes .....	6-25
6-13	Server Overview Editor Attributes .....	6-28
7-1	Oracle Event Processing Type Dialog.....	7-16
7-2	EPN Editor Icons.....	7-18
7-3	New Adapter Wizard - Page 1 .....	7-22
7-4	New Processor Dialog .....	7-27
9-1	Data Types for Event Types .....	9-4
9-2	csvgen Adapter Types.....	9-5
9-3	EPN Assembly File event-type Element Property Attributes .....	9-5
9-4	SQL Column Types and Oracle Event Processing Type Equivalents .....	9-6
11-1	jms-adapter Inbound Child Elements.....	11-20
11-2	jms-adapter Outbound Component Configuration Child Elements.....	11-22
12-1	http-pub-sub-adapter for Publishing Component Configuration Child Elements.....	12-14
12-2	http-pub-sub-adapter for Subscribing Component Configuration Child Elements...	12-15
14-1	bea-jaxws.xml File Attributes.....	14-3
15-1	Interfaces to Support Suspending and Resuming an Adapter.....	15-8
16-1	Interfaces for Implementing an Event Source .....	16-7
16-2	Interfaces Implemented by Sender Classes.....	16-7
16-3	Comparison of Event Beans and Spring Beans .....	16-10
17-1	EPN Assembly File table Element Attributes .....	17-9
18-1	spatial:context Element Attributes .....	18-2
20-1	Child Elements of bdb-config.....	20-5
20-2	Child Elements of record-parameters .....	20-6
20-3	Child Elements of playback-parameters .....	20-9
21-1	Load Generator Properties .....	21-3

22-1	Event Inspector JSON Event Required Attributes .....	22-3
23-1	Oracle Event Processing Application Library Path .....	23-15
23-2	Oracle Event Processing Application Library Path Variable .....	23-16
23-3	bundler.sh Command Line Options .....	23-19
23-4	Factory Class and Service Interfaces .....	23-20
23-5	New Java Class Parameters .....	23-22
23-6	weblogic.i18ngen Utility Options .....	23-37
24-1	Oracle Event Processing High Availability Quality of Service .....	24-9
24-2	Oracle Event Processing High Availability Application Types .....	24-14
24-3	Child Elements of wlevs:adapter for the High Availability Input Adapter.....	24-45
24-4	High Availability Input Adapter Instance Properties .....	24-45
24-5	Child Elements of ha-inbound-adapter for the High Availability Input Adapter.....	24-46
24-6	Child Elements of wlevs:adapter for the Buffering Output Adapter.....	24-46
24-7	Buffering Output Adapter Instance Properties .....	24-47
24-8	Child Elements of ha-buffering-adapter for the Buffering Output Adapter .....	24-47
24-9	Child Elements of wlevs:adapter for the Broadcast Output Adapter .....	24-48
24-10	Broadcast Output Adapter Instance Properties .....	24-48
24-11	Child Elements of ha-broadcast-adapter for the Broadcast Output Adapter .....	24-49
24-12	Child Elements of wlevs:adapter for the Correlating Output Adapter .....	24-50
24-13	Correlating Output Adapter Instance Properties.....	24-50
24-14	Child Elements of ha-correlating-adapter for the Correlating Output Adapter .....	24-50
25-1	Event Partitioner Channel Threading Options.....	25-3
25-2	New Java Class Options for EventPartitioner .....	25-11
25-3	Oracle Event Processing Server Configuration File groups Element Configuration..	25-15
25-4	Oracle Event Processing Server Configuration File groups Element Configuration..	25-20
C-1	Attributes of the wlevs:adapter Application Assembly Element .....	C-4
C-2	Attributes of the wlevs:application-timestamped Application Assembly Element .....	C-6
C-3	Attributes of the wlevs:cache Application Assembly Element .....	C-7
C-4	Attributes of the wlevs:cache-listener Application Assembly Element.....	C-8
C-5	Attributes of the wlevs:cache-loader Application Assembly Element .....	C-8
C-6	Attributes of the wlevs:cache-source Application Assembly Element .....	C-9
C-7	Attributes of the wlevs:cache-store Application Assembly Element .....	C-10
C-8	Attributes of the wlevs:caching-system Application Assembly Element.....	C-11
C-9	Attributes of the wlevs:channel Application Assembly Element .....	C-12
C-10	Attributes of the wlevs:event-bean Application Assembly Element .....	C-14
C-11	Attributes of the wlevs:event-type Application Assembly Element .....	C-16
C-12	Attributes of the wlevs:factory Application Assembly Element .....	C-18
C-13	Attributes of the wlevs:function Application Assembly Element .....	C-19
C-14	Attributes of the wlevs:instance-property Application Assembly Element .....	C-26
C-15	Attributes of the wlevs:listener Application Assembly Element.....	C-27
C-16	Attributes of the wlevs:metadata Application Assembly Element .....	C-28
C-17	Attributes of the wlevs:processor Application Assembly Element.....	C-28
C-18	Attributes of the wlevs:properties Application Assembly Element.....	C-29
C-19	Attributes of the wlevs:property Application Assembly Element .....	C-30
C-20	Attributes of the wlevs:property Application Assembly Element .....	C-32
C-21	Attributes of the wlevs:source Application Assembly Element .....	C-33
C-22	Attributes of the wlevs:table Application Assembly Element .....	C-33
C-23	Attributes of the wlevs:table-source Application Assembly Element .....	C-34
D-1	Attributes of the binding Component Configuration Element.....	D-21
D-2	Attributes of the database Component Configuration Element .....	D-35
D-3	Attributes of the group-binding Component Configuration Element.....	D-48
D-4	Attributes of the listeners Component Configuration Element .....	D-54
D-5	Attributes of the param Component Configuration Element .....	D-60
D-6	Attributes of the params Component Configuration Element.....	D-62
D-7	Attributes of the query Component Configuration Element .....	D-70

D-8	Attributes of the rule Component Configuration Element.....	D-73
D-9	Attributes of the view Component Configuration Element .....	D-93
E-1	Attributes of the wlevs:deployment Deployment Element .....	E-2
F-1	Child Elements of: auth-constraint.....	F-5
F-2	Child Elements of: bdb-config.....	F-6
F-3	Child Elements of: channel-resource-collection .....	F-9
F-4	Child Elements of: cluster.....	F-10
F-5	Child Elements of: connection-pool-params.....	F-12
F-6	Child Elements of: data-source-params.....	F-16
F-7	Child Elements of: driver-params .....	F-18
F-8	Child Elements of: debug.....	F-20
F-9	Child Elements of: event-store .....	F-20
F-10	Child Elements of: exported-jndi-context .....	F-21
F-11	Child Elements of: jetty .....	F-23
F-12	Child Elements of: jetty-web-app .....	F-24
F-13	Child Elements of: jmx .....	F-25
F-14	Child Elements of: jndi-context.....	F-25
F-15	Child Elements of: log-file .....	F-26
F-16	Child Elements of: log-stdout .....	F-27
F-17	Child Elements of: logging-service.....	F-28
F-18	Child Elements of: netio.....	F-31
F-19	Child Elements of: netio-client.....	F-31
F-20	Child Elements of: rdbms-event-store-provider .....	F-34
F-21	Child Elements of: rmi .....	F-35
F-22	Child Elements of: scheduler .....	F-36
F-23	Child Elements of: server-config .....	F-37
F-24	Child Elements of: services.....	F-39
F-25	Child Elements of: show-detail-error-message .....	F-40
F-26	Child Elements of: ssl .....	F-41
F-27	Child Elements of: transaction-manager .....	F-43
F-28	Child Elements of: use-secure-connections.....	F-46
F-29	Child Elements of: weblogic-instances .....	F-46
F-30	Child Elements of: weblogic-rmi-client .....	F-48
F-31	Child Elements of: work-manager .....	F-49
F-32	Child Elements of: xa-params .....	F-49
G-1	Attributes of the message_catalog Element .....	G-3
G-2	Attributes of the logmessage Element .....	G-5
G-3	Attributes of the message Element.....	G-7
H-1	Attributes of the locale_message_catalog Element .....	H-3
H-2	Attributes of the logmessage Element .....	H-4
H-3	Attributes of the message Element.....	H-5
I-1	Attributes of the com.bea.wlevs.util.Service JWS Annotation Tag .....	I-6



## List of Examples

1-1	EPN Assembly File With Nested Bean .....	1-11
1-2	EPN Assembly File With all Nodes Nested .....	1-11
1-3	Application 1 Referencing Foreign Stage in Application 2.....	1-12
1-4	Foreign Stage in Application 2.....	1-12
1-5	Adding a ConfigurationPropertyPlaceholderConfigurer.....	1-13
1-6	Sample Resource: Data Source StockDS .....	1-14
1-7	Static Resource Injection Using Static Resource Names: Annotations.....	1-15
1-8	Static Resource Injection Using Static Resource Names: XML.....	1-15
1-9	Static Resource Injection Using Static Resource Names: Annotations.....	1-15
1-10	Static Resource Injection Using Static Resource Names: XML.....	1-16
1-11	Custom Component Configuration .....	1-16
1-12	Static Resource Injection Using Dynamic Resource Names: Annotations .....	1-16
1-13	Static Resource Injection Using Dynamic Resource Names: XML .....	1-16
1-14	Dynamic Resource Injection: Annotations .....	1-16
1-15	Dynamic Resource Lookup Using JNDI.....	1-17
2-1	replay Application Configuration File config.xml: adapter Element.....	2-93
4-1	Default eclipse.ini File .....	4-11
4-2	Memory Resources .....	4-11
4-3	Virtual Machine Path.....	4-12
5-1	Accessing a Properties File .....	5-33
7-1	Assembly Source for EPN With Nested Bean.....	7-9
7-2	Assembly Source for EPN With all Nodes Nested .....	7-10
8-1	GetHighVolume Query Element with CQL Code .....	8-23
9-1	Event Type Repository .....	9-7
9-2	MarketEvent Class .....	9-8
9-3	MarketEvent Class .....	9-11
9-4	EPN Assembly File event-type-repository .....	9-12
9-5	Programmatically Registering an Event.....	9-12
9-6	Specifying com.bea.welvs.ede.api.Type Data Types for Tuple Event Type Properties	9-15
9-7	EPN Assembly File event-type-repository .....	9-17
9-8	Specifying Java Data Types for java.util.Map Event Type Properties .....	9-19
9-9	EPN Assembly File event-type-repository .....	9-19
9-10	Programmatically Registering an Event.....	9-20
9-11	EPN Assembly File With OSGi Reference to EventTypeRepository.....	9-21
9-12	Accessing the EventTypeRepository in the MyBean Implementation.....	9-21
9-13	Java Source File Using the @ServiceReference Annotation .....	9-21
9-14	Java Source File Using the @Service Annotation .....	9-21
10-1	EPN Assembly File Channel Id: priceStream .....	10-1
10-2	Component Configuration File Channel Name: priceStream .....	10-1
10-3	Channel as Relation: primary-key Attribute.....	10-3
10-4	PriceEvent .....	10-4
10-5	filterFanoutProcessor Oracle CQL Queries.....	10-5
10-6	Using selector to Control Which Query Results are Output .....	10-5
10-7	Batch Processing Channel.....	10-6
10-8	Component Configuration File Header and config Element .....	10-8
10-9	Component Configuration File Channel Element .....	10-8
10-10	EPN Assembly File Channel Id: priceStream .....	10-9
10-11	Component Configuration File Channel Name: priceStream .....	10-9
10-12	Component Configuration File Header and config Element .....	10-12
10-13	Component Configuration File Channel Element .....	10-12
10-14	EPN Assembly File Channel Id: priceStream .....	10-13
10-15	Component Configuration File Channel Name: priceStream .....	10-13
10-16	Sample Channel Component Configuration File .....	10-17
10-17	Channel EPN Assembly File .....	10-18



11-1	wlevs:adapter Element for Inbound Adapter.....	11-5
11-2	jms-adapter Element for Inbound Adapter.....	11-6
11-3	jms-adapter Element With Tibco EMS JMS Configuration.....	11-6
11-4	wlevs:adapter Element for Inbound Adapter.....	11-8
11-5	jms-adapter Element for Inbound Adapter.....	11-8
11-6	jms-adapter Elements for an Oracle WebLogic Server JMS Provider.....	11-9
11-7	wlevs:adapter Element for Inbound Adapter.....	11-11
11-8	jms-adapter Element for Inbound Adapter.....	11-11
11-9	jms-adapter Element With Tibco EMS JMS Configuration.....	11-11
11-10	Custom Converter for an Inbound JMS Adapter.....	11-13
11-11	Specifying a Converter Class for an Inbound JMS Adapter in the EPN Assembly File.....	11-14
11-12	Custom Converter for an Outbound JMS Adapter.....	11-15
11-13	Specifying a Converter Class for an Outbound JMS Adapter in the EPN Assembly File.....	11-15
13-1	Component Configuration File Name Values.....	13-4
13-2	EPN Assembly File ID and Ref Values.....	13-4
13-3	Component Configuration File: Coherence Cache.....	13-8
13-4	Oracle Coherence Cache LocalListener Implementation.....	13-19
13-5	Oracle Coherence Cache EPN Assembly File for a Cache Loader.....	13-21
13-6	Oracle Coherence Cache LocalLoader Implementation.....	13-21
13-7	Oracle Coherence Cache EPN Assembly File for a Cache Store.....	13-22
13-8	Oracle Coherence Cache LocalStore Implementation.....	13-22
13-9	Valid Oracle CQL Query Against a Cache.....	13-23
14-1	bea-jaxws.xml File.....	14-3
15-1	High-Level View of Input Adapter Class.....	15-4
16-1	Implementing the StreamSink Interface.....	16-5
16-2	Implementing the RelationSink Interface.....	16-6
16-3	Implementing the RelationSource Interface.....	16-8
17-1	EPN Assembly File Oracle CQL Processor Id: proc.....	17-2
17-2	Component Configuration File Oracle CQL Processor Name: proc.....	17-2
17-3	Default Processor Component Configuration.....	17-4
17-4	Table Create SQL Statement.....	17-7
17-5	Oracle CQL Query on Relational Database Table Stock.....	17-8
17-6	Oracle Event Processing Server config.xml File With Data Source StockDS.....	17-8
17-7	EPN Assembly File table Element.....	17-9
17-8	EPN Assembly File table-source Element.....	17-10
17-9	EPN Assembly File event-type element for a Table.....	17-10
17-10	Oracle CQL Query Using Table Event Type StockEvent.....	17-10
17-11	Query Configured to Allow Parallel Execution.....	17-12
17-12	Query Configured to Allow Parallel Execution Across Partitions.....	17-12
17-13	Fault Handler Class.....	17-16
17-14	Code to Register a Fault Handler with an EPN Stage.....	17-17
18-1	EPN Assembly File: Oracle Spatial Namespace and Schema Location.....	18-2
18-2	spatial:context Element in EPN Assembly File.....	18-2
18-3	spatial:context Element in EPN Assembly File.....	18-3
18-4	Referencing spatial:context in an Oracle CQL Query.....	18-3
18-5	EPN Assembly File: Oracle JDBC Data Cartridge Namespace and Schema Location..	18-4
18-6	jdbc:jdbc-context Element in EPN Assembly File: id.....	18-4
18-7	Component Configuration File: Oracle JDBC Data Cartridge Namespace.....	18-5
18-8	jc:jdbc-ctx Element in Component Configuration File.....	18-5
18-9	jc:jdbc-ctx Element in Component Configuration File: name.....	18-5
18-10	jc:jdbc-ctx Element in Component Configuration File: data-source.....	18-5
18-11	jc:jdbc-ctx Element in Component Configuration File: function.....	18-5
18-12	Referencing JDBC Application Context in an Oracle CQL Query.....	18-6

19-1	EPN Assembly File EPL Processor Id: proc .....	19-2
19-2	Component Configuration File EPL Processor Name: proc .....	19-2
20-1	bdb-config Element.....	20-5
20-2	Default bdb-config Element .....	20-11
21-1	EmployeeEvent Event Type .....	21-3
21-2	Data Feed File for EmployeeEvent Event Type.....	21-3
22-1	Event Inspector JSON Event.....	22-3
22-2	Event Inspector Service Local HTTP Pub-Sub Server .....	22-5
22-3	Oracle Event Processing Built-In HTTP Pub-Sub Server http-pubsub Element .....	22-5
22-4	Event Inspector Service Remote HTTP Pub-Sub Server .....	22-6
22-5	Oracle Event Processing Built-In HTTP Pub-Sub Server http-pubsub Element .....	22-6
22-6	Event Injection Component Configuration Settings.....	22-7
22-7	Event Tracing Component Configuration Settings.....	22-8
23-1	bundler.sh Command Line Options .....	23-19
23-2	Using the Bundler Utility.....	23-20
23-3	Bundle JAR Contents.....	23-20
23-4	Service Registration Log Messages .....	23-20
23-5	MyActivator Class Implementation .....	23-22
23-6	Un-JAR the Database Driver .....	23-28
23-7	Adding Export-Package to the Manifest Editor .....	23-28
23-8	Adding a Bundle-Activator Element to the Manifest Editor.....	23-29
23-9	Adding a DynamicImport-Package Element to the Manifest Editor .....	23-29
23-10	Message Arguments .....	23-34
23-11	Log Message Catalog.....	23-35
23-12	Simple Text Catalog.....	23-35
23-13	Locale-Specific Catalog .....	23-36
24-1	Simple Failover EPN Assembly File .....	24-20
24-2	Simple Failover Component Configuration Assembly File.....	24-20
24-3	Simple Failover EPN Assembly File: Buffering Output Adapter .....	24-21
24-4	Application Timestamp Configuration .....	24-22
24-5	Configuring windowLength in the Buffering Output Adapter.....	24-22
24-6	Simple Failover Component Configuration File With High Availability Adapters....	24-23
24-7	Simple Failover With Buffering EPN Assembly File .....	24-23
24-8	Simple Failover With Buffering Component Configuration Assembly File .....	24-24
24-9	Simple Failover EPN Assembly File: Buffering Output Adapter .....	24-25
24-10	Application Timestamp Configuration .....	24-26
24-11	Configuring windowLength in the Buffering Output Adapter.....	24-26
24-12	Simple Failover With Buffering Component Configuration File.....	24-26
24-13	Light-Weight Queue Trimming EPN Assembly File.....	24-27
24-14	Light-Weight Queue Trimming Component Configuration Assembly File.....	24-28
24-15	Light-Weight Queue Trimming EPN Assembly File: High Availability Input Adapter .....	24-28
24-16	Light-Weight Queue Trimming EPN Assembly File: Broadcast Output Adapter.....	24-29
24-17	High Availability Input Adapter: Default Configuration.....	24-30
24-18	High Availability Input Adapter: Tuple Events.....	24-30
24-19	High Availability Input Adapter: Key of One Event Property .....	24-31
24-20	High Availability Input Adapter: Key of Multiple Event Properties.....	24-31
24-21	MyCompoundKeyClass Implementation .....	24-31
24-22	Application Timestamp Configuration .....	24-32
24-23	Broadcast Output Adapter: Default Configuration .....	24-32
24-24	Broadcast Output Adapter: Key of One Event Property .....	24-32
24-25	Broadcast Output Adapter: Key of Multiple Event Properties .....	24-33
24-26	MyCompoundKeyClass Implementation .....	24-33
24-27	Light-Weight Queue Trimming Component Configuration File .....	24-34
24-28	Precise Recovery With JMS EPN Assembly File .....	24-35

24-29	Precise Recovery With JMS Component Configuration Assembly File .....	24-36
24-30	Precise Recovery With JMS EPN Assembly File: High Availability Input Adapter....	24-37
24-31	Precise Recovery With JMS EPN Assembly File: Correlating Output Adapter .....	24-37
24-32	High Availability Input Adapter: Default Configuration.....	24-38
24-33	High Availability Input Adapter: Tuple Events.....	24-39
24-34	High Availability Input Adapter: Key of One Event Property .....	24-39
24-35	High Availability Input Adapter: Key of Multiple Event Properties.....	24-39
24-36	MyCompoundKeyClass Implementation .....	24-39
24-37	Application Timestamp Configuration .....	24-40
24-38	Correlating Output Adapter Configuration: failOverDelay .....	24-40
24-39	Inbound JMS Adapter Assembly File .....	24-41
24-40	Inbound JMS Adapter Component Configuration File.....	24-41
24-41	Creating the Correlated Source.....	24-41
24-42	Correlating Output Adapter: correlatedSource.....	24-42
24-43	Inbound and Outbound JMS Adapter Component Configuration File .....	24-42
24-44	High Availability Input and Output Adapter Component Configuration File.....	24-43
24-45	High Availability Input Adapter EPN Assembly File.....	24-44
24-46	High Availability Input Adapter Component Configuration File .....	24-45
24-47	Buffering Output Adapter EPN Assembly File.....	24-46
24-48	Buffering Output Adapter Component Configuration File .....	24-47
24-49	Broadcast Output Adapter EPN Assembly File .....	24-48
24-50	Broadcast Output Adapter Component Configuration File.....	24-49
24-51	Correlating Output Adapter EPN Assembly File .....	24-49
24-52	Correlating Output Adapter Component Configuration File .....	24-50
25-1	ActiveActiveGroupBean bean Element .....	25-4
25-2	Common jms-adapter Selector Definitions .....	25-5
25-3	Definition of Event Type PriceEvent.....	25-7
25-4	EventPartitioner Class.....	25-11
25-5	EventPartitioner Class Implementation .....	25-12
25-6	ActiveActiveGroupBean bean Element .....	25-16
25-7	jms-adapter Selector Definition for ocep-server-1 .....	25-16
25-8	Precise Recovery With JMS EPN Assembly File .....	25-17
25-9	Precise Recovery With JMS Component Configuration Assembly File .....	25-18
25-10	ActiveActiveGroupBean bean Element .....	25-21
25-11	jms-adapter Element for Inbound JMS Adapters.....	25-21
25-12	jms-adapter Selector Definition for ocep-server-1 .....	25-21
25-13	jms-adapter Element for Outbound JMS Adapters.....	25-22
25-14	ActiveActiveGroupBean bean Element With groupPattern Attribute .....	25-23
26-1	Annotated Custom Adapter Implementation .....	26-3
26-2	Extended Component Configuration: Annotations.....	26-4
26-3	Extended Component Configuration File: XSD .....	26-6
26-4	Custom Adapter Implementation .....	26-9
26-5	Extended Component Configuration.....	26-10
C-1	Single-Row User Defined Function Implementation Class .....	C-20
C-2	Single-Row User Defined Function for an Oracle CQL Processor .....	C-20
C-3	Invoking the Single-Row User-Defined Function on an Oracle CQL Processor .....	C-20
C-4	Single-Row User Defined Function Implementation Class .....	C-21
C-5	Single-Row User Defined Function for an EPL Processor .....	C-21
C-6	Invoking the Single-Row User-Defined Function on an EPL Processor.....	C-21
C-7	Aggregate User Defined Function Implementation Class.....	C-21
C-8	Aggregate User Defined Function for an Oracle CQL Processor .....	C-23
C-9	Invoking the Aggregate User-Defined Function on an Oracle CQL Processor .....	C-23
C-10	Aggregate User Defined Function Implementation Class.....	C-23
C-11	Aggregate User Defined Function for an EPL Processor .....	C-24
C-12	Invoking the Aggregate User-Defined Function on an EPL Processor.....	C-24

C-13	User Defined Function Using Nested Bean Element.....	C-25
C-14	User Defined Function Using Reference .....	C-25
D-1	adapter Element Hierarchy .....	D-5
D-2	http-pub-sub-adapter Element Hierarchy.....	D-5
D-3	jms-adapter Element Hierarchy .....	D-6
D-4	processor (EPL) Element Hierarchy .....	D-8
D-5	processor (Oracle CQL) Element Hierarchy .....	D-9
D-6	stream Element Hierarchy .....	D-10
D-7	channel Element Hierarchy .....	D-10
D-8	event-bean Element Hierarchy .....	D-11
D-9	caching-system Element Hierarchy .....	D-12
D-10	coherence-caching-system Element Hierarchy.....	D-13
D-11	diagnostic-profiles Element Hierarchy .....	D-13
D-12	filterFanoutProcessor Oracle CQL Queries.....	D-76
D-13	Using selector to Control Which Query Results are Output .....	D-76
G-1	Log Message Catalog Hierarchy.....	G-1
G-2	Simple Text Catalog Hierarchy .....	G-2
G-3	Log Message Catalog.....	G-2
G-4	Simple Text Catalog.....	G-2
H-1	Locale-Specific Log Message Catalog Hierarchy .....	H-1
H-2	Locale-Specific Simple Text Catalog Hierarchy .....	H-2
H-3	Locale-Specific Log Message Catalog .....	H-2
H-4	Locale-Specific Simple Text Catalog .....	H-2
I-1	@Activate Annotation .....	I-2
I-2	HelloWorldAdapterConfig.....	I-3
I-3	@Prepare Annotation .....	I-4
I-4	@Rollback Annotation.....	I-5
I-5	@Service Annotation.....	I-7

## List of Figures

1-1	Oracle Event Processing Application Lifecycle State Diagram.....	1-21
2-1	The HelloWorld Example Event Processing Network.....	2-4
2-2	The CQL Example Event Processing Network.....	2-8
2-3	Oracle Event Processing Visualizer Logon Screen.....	2-13
2-4	Oracle Event Processing Visualizer Dashboard.....	2-14
2-5	CQL Application Screen: General Tab.....	2-15
2-6	CQL Application: Event Processing Network Tab.....	2-16
2-7	Oracle CQL Processor: General Tab.....	2-17
2-8	Oracle CQL Processor: Query Wizard Tab.....	2-18
2-9	Template Tab.....	2-19
2-10	SSource Configuration Dialog.....	2-20
2-11	Pattern Configuration Dialog: Pattern Tab.....	2-21
2-12	Pattern Configuration Dialog: Define Tab.....	2-22
2-13	Expression Builder: CustOrder.....	2-23
2-14	Pattern Configuration Dialog: Define Tab With CustOrder Condition.....	2-24
2-15	Expression Builder: NoApproval.....	2-25
2-16	Expression Builder: Shipment.....	2-26
2-17	Pattern Configuration Dialog: Define Tab Complete.....	2-27
2-18	Measure Tab.....	2-28
2-19	Expression Builder: orderid.....	2-29
2-20	Expression Builder: amount.....	2-30
2-21	Measure Tab: Complete.....	2-31
2-22	Select Configuration Dialog: Project Tab.....	2-32
2-23	Select Configuration Dialog: Project Tab Complete.....	2-33
2-24	Output Configuration Dialog.....	2-34
2-25	Inject Rule Confirmation Dialog.....	2-34
2-26	CQL Rules Tab With Tracking Query.....	2-35
2-27	Stream Visualizer: Showing Missing Events.....	2-36
2-28	Oracle Event Processing Visualizer Logon Screen.....	2-37
2-29	Oracle Event Processing Visualizer Dashboard.....	2-38
2-30	CQL Application Screen: General Tab.....	2-39
2-31	CQL Application: Event Processing Network Tab.....	2-39
2-32	Oracle CQL Processor: General Tab.....	2-40
2-33	Oracle CQL Processor: Query Wizard Tab.....	2-40
2-34	Query Wizard: SSource.....	2-41
2-35	SSource Configuration Dialog.....	2-42
2-36	Query Wizard: Filter.....	2-43
2-37	Connecting the SSource and Filter Icons.....	2-43
2-38	Filter Configuration Dialog.....	2-44
2-39	Filter Expression Builder.....	2-45
2-40	Filter Configuration Dialog: After Adding the Filter.....	2-46
2-41	Query Wizard: Select.....	2-47
2-42	Select Configuration Dialog: Properties Selected.....	2-48
2-43	Query Wizard: Output.....	2-49
2-44	Output Configuration Dialog.....	2-50
2-45	Inject Rule Confirmation Dialog.....	2-50
2-46	CQL Rules Tab With View StockVolGt1000.....	2-51
2-47	Oracle Event Processing Visualizer Logon Screen.....	2-52
2-48	Oracle Event Processing Visualizer Dashboard.....	2-53
2-49	CQL Application Screen: General Tab.....	2-54
2-50	CQL Application: Event Processing Network Tab.....	2-54
2-51	Oracle CQL Processor: General Tab.....	2-55
2-52	Oracle CQL Processor: Query Wizard Tab.....	2-55
2-53	Query Wizard: SSource for Moving Average Query.....	2-56

2-54	SSource Configuration Dialog: Moving Average Query .....	2-57
2-55	Query Wizard: Window for Moving Average Query .....	2-58
2-56	Window Configuration Dialog: After Adding Window .....	2-59
2-57	Query Wizard: Select for Moving Average Query.....	2-60
2-58	Select Configuration Dialog: Source Property symbol Selected .....	2-61
2-59	Select Configuration Dialog: Source Property symbol Mapped to Output Event Property....	2-62
2-60	Select Configuration Dialog: Source Property price Selected .....	2-63
2-61	Expression Builder: Applying the AVG Function.....	2-64
2-62	Select Configuration Dialog: With Expression .....	2-65
2-63	Select Configuration Dialog: Source Property price Mapped to Output Event Property .....	2-66
2-64	Validation Error: GROUP BY .....	2-66
2-65	Group Tab: With symbol Grouping Property .....	2-67
2-66	Query Wizard: Output .....	2-68
2-67	Output Configuration Dialog.....	2-69
2-68	Inject Rule Confirmation Dialog .....	2-69
2-69	CQL Rules Tab With View MovingAverage .....	2-70
2-70	Stream Visualizer: Showing Moving Average Query Output .....	2-71
2-71	Oracle Spatial Example Event Processing Network .....	2-72
2-72	Oracle Spatial Web Page .....	2-74
2-73	Oracle Spatial Web Page: Bus Stop Arrivals Tab .....	2-75
2-74	Oracle Spatial Web Page: Bus Tracking.....	2-76
2-75	FX Example Event Processing Network.....	2-79
2-76	The Signal Generation Example Event Processing Network .....	2-84
2-77	Signal Generation Dashboard .....	2-86
2-78	The Event Record and Playback Example Event Processing Network.....	2-89
2-79	Oracle Event Processing Visualizer Logon Screen.....	2-91
2-80	Oracle Event Processing Visualizer Dashboard .....	2-92
2-81	Event Record Tab .....	2-93
2-82	Start Recording Alert Dialog .....	2-94
2-83	Event Playback Tab.....	2-94
2-84	Start Playback Alert Dialog .....	2-95
2-85	Stream Visualizer .....	2-96
4-1	Install Dialog.....	4-3
4-2	Add Site Dialog .....	4-3
4-3	Install Dialog - Site Selected .....	4-4
4-4	Install Dialog - Install Details .....	4-5
4-5	About Eclipse.....	4-5
4-6	About Eclipse Features Dialog.....	4-6
4-7	Feature Plug-ins Dialog.....	4-6
4-8	Install Dialog.....	4-8
4-9	Add Site Dialog .....	4-8
4-10	Select Local Site Archive Dialog .....	4-9
4-11	About Eclipse.....	4-9
4-12	About Eclipse Features Dialog.....	4-10
4-13	Feature Plug-ins Dialog.....	4-10
4-14	Configuration Details for Java 6 .....	4-13
5-1	Oracle Event Processing Project Structure .....	5-2
5-2	New Project - Select a Wizard Dialog .....	5-3
5-3	New Oracle Event Processing Application Project Wizard: Create an Oracle Event Processing Application	5-4
5-4	New Oracle Event Processing Application Project Wizard: Oracle Event Processing Application Content	5-5
5-5	New Oracle Event Processing Application Project Wizard: Template Dialog .....	5-6

5-6	New Dialog .....	5-7
5-7	New OEP Assembly File Dialog .....	5-8
5-8	New Dialog .....	5-9
5-9	New OEP Application Configuration File Dialog .....	5-10
5-10	Oracle Event Processing Project build.properties File.....	5-11
5-11	Oracle Event Processing Applications Export: Select Project Dialog .....	5-12
5-12	Project Properties Dialog: Project Facets .....	5-14
5-13	Modify Faceted Project.....	5-15
5-14	Preferences Dialog .....	5-16
5-15	Project Properties Dialog: Targeted Runtimes.....	5-17
5-16	Builder Error .....	5-18
5-17	Import Projects Dialog .....	5-19
5-18	Project Properties Dialog: Project Facets .....	5-20
5-19	Modify Faceted Project.....	5-20
5-20	Preferences Dialog .....	5-21
5-21	Project Properties Dialog: Targeted Runtimes.....	5-22
5-22	Builder Error .....	5-23
5-23	Preferences Dialog .....	5-24
5-24	Oracle Event Processing IDE for Eclipse lib Directory .....	5-26
5-25	Manifest Editor: Build Tab.....	5-27
5-26	Manifest Editor - Runtime Tab .....	5-28
5-27	JAR Selection Dialog.....	5-28
5-28	Manifest Editor Runtime tab After Adding a JAR to the Classpath .....	5-29
5-29	Manifest Editor MANIFEST.MF Tab .....	5-30
5-30	Package Explorer.....	5-30
5-31	Manifest Editor: Dependencies Tab .....	5-31
5-32	Plug-in Selection Dialog .....	5-32
5-33	Manifest Editor: Build Tab.....	5-33
5-34	Oracle Event Processing IDE for Eclipse lib Directory .....	5-34
5-35	Manifest Editor: Runtime tab .....	5-35
5-36	Package Selection Dialog .....	5-35
5-37	Manifest Editor Runtime tab After Exporting a Package .....	5-36
5-38	Oracle Event Processing IDE for Eclipse lib Directory .....	5-37
5-39	Manifest Editor: Dependencies tab .....	5-37
5-40	Package Selection Dialog .....	5-38
5-41	Manifest Editor Dependencies tab After Importing a Package.....	5-38
5-42	Oracle Event Processing Problem Severities Dialog: Workspace .....	5-40
6-1	Oracle Event Processing IDE for Eclipse Server View .....	6-4
6-2	New Server: Define New Server Dialog (No Installed Runtimes) .....	6-5
6-3	New Server: New Oracle Event Processing v11.1 Runtime Dialog.....	6-6
6-4	New Server: Define New Server (Installed Runtimes) Dialog.....	6-7
6-5	New Server: New Oracle Event Processing v11.1 Server.....	6-8
6-6	New Server: New Oracle Event Processing v11 Server Dialog for a Local Server.....	6-9
6-7	Oracle Event Processing IDE for Eclipse Server View .....	6-11
6-8	New Server: Define New Server Dialog (No Installed Runtimes) .....	6-12
6-9	New Server: New Oracle Event Processing v11.1 Runtime Dialog.....	6-13
6-10	New Server: Define New Server (Installed Runtimes) Dialog.....	6-14
6-11	New Server: New Oracle Event Processing v11.1 Server.....	6-15
6-12	New Server: New Oracle Event Processing v11 Server Dialog for a Remote Server....	6-16
6-13	Preferences - Server - Installed Runtimes.....	6-17
6-14	New Server Runtime Environment Dialog .....	6-18
6-15	New Server Runtime Environment: New Oracle Event Processing v11.1 Runtime Dialog ....	6-19
6-16	Starting an Oracle Event Processing Server .....	6-21
6-17	Stopping an Oracle Event Processing Server.....	6-21

6-18	Attaching to an Existing Local Oracle Event Processing Server Instance .....	6-22
6-19	Attaching to an Existing Remote Oracle Event Processing Server Instance .....	6-23
6-20	Stopping an Oracle Event Processing Server.....	6-24
6-21	Adding a Project to an Oracle Event Processing Server.....	6-25
6-22	Add and Remove Dialog .....	6-25
6-23	Server View After Adding a Project.....	6-26
6-24	Select Cluster Deployment Group Name Dialog .....	6-26
6-25	Server View After Deploying (Publishing) a Project .....	6-27
6-26	Server Overview Editor .....	6-27
6-27	Editing the Domain Configuration File .....	6-31
6-28	Oracle Event Processing Domain Configuration File config.xml .....	6-31
6-29	Opening the Oracle Event Processing Visualizer.....	6-32
6-30	Oracle Event Processing Visualizer.....	6-33
6-31	Setting a Breakpoint.....	6-34
6-32	Starting the Oracle Event Processing Server in Debug Mode .....	6-34
7-1	Opening the EPN Editor from a Project .....	7-2
7-2	EPN Editor .....	7-2
7-3	Opening the EPN Editor from a Context or Configuration File .....	7-3
7-4	EPN Editor .....	7-4
7-5	EPN Flow Representation.....	7-5
7-6	Filtering the EPN by Assembly File .....	7-5
7-7	Zoom Level .....	7-6
7-8	Optimize Layout .....	7-6
7-9	Show /Hide Unconnected Beans .....	7-7
7-10	Exporting the EPN as an Image File.....	7-7
7-11	Printing the EPN .....	7-7
7-12	Configuration Badging.....	7-8
7-13	Link Source .....	7-8
7-14	Link Source Assembly File .....	7-8
7-15	Link Listener .....	7-8
7-16	Link Listener Assembly File .....	7-9
7-17	EPN With Nested Bean .....	7-9
7-18	EPN With all Nodes Nested .....	7-10
7-19	Event Type Repository Editor.....	7-10
7-20	Node with Configuration Badge .....	7-11
7-21	Component Configuration File: Hyperlinking to EPN Assembly File .....	7-12
7-22	EPN Assembly File: Hyperlinking to Component Configuration File .....	7-13
7-23	Oracle CQL Statement: Event Schema.....	7-13
7-24	Corresponding Event Definition in EPN Assembly File.....	7-14
7-25	Example Oracle Event Processing EPN .....	7-15
7-26	Oracle Event Processing Type Browser.....	7-16
7-27	Opening the FilterAsia EPN Assembly File.....	7-17
7-28	Opening the FilterAsia Component Configuration File.....	7-17
7-29	Creating a Basic Node .....	7-20
7-30	New Basic Node.....	7-20
7-31	Creating an Adapter Node .....	7-21
7-32	New Adapter Wizard .....	7-21
7-33	New Adapter Wizard - jms-inbound .....	7-23
7-34	New Adapter Wizard - jms-outbound.....	7-24
7-35	New Adapter Wizard - httppub .....	7-24
7-36	New Adapter Wizard - httpsub .....	7-25
7-37	New Adapter Node .....	7-25
7-38	Creating a Processor Node .....	7-26
7-39	New Processor Dialog.....	7-27
7-40	New Processor Node.....	7-27



7-41	Connecting Nodes: Connection Allowed.....	7-28
7-42	Connecting Nodes: Connection Forbidden.....	7-29
7-43	Valid Connections.....	7-29
7-44	EPN Assembly File: Before Connection.....	7-29
7-45	EPN Assembly File: After Connection.....	7-30
7-46	Laying Out Nodes.....	7-30
7-47	Renaming Nodes.....	7-30
7-48	Deleting Nodes.....	7-31
7-49	EPN Before Deleting a Channel Node.....	7-31
7-50	Assembly File Before Deleting a Channel Node.....	7-31
7-51	EPN After Deleting a Channel Node.....	7-32
7-52	Assembly File After Deleting a Channel Node.....	7-32
9-1	Event Type Repository Editor - JavaBean Event.....	9-10
9-2	Event Type Repository Editor - Tuple Event.....	9-16
10-1	EPN With Oracle CQL Processor and Down-Stream Channel.....	10-5
10-2	Channel With Configuration Badge.....	10-10
10-3	Channel With Configuration Badge.....	10-14
10-4	EPN with Two Channels.....	10-17
12-1	Built-In Pub-Sub Adapter For Local Publishing.....	12-3
12-2	Built-In Pub-Sub Adapter For Remote Publishing.....	12-3
12-3	Built-In Pub-Sub Adapter For Subscribing.....	12-4
13-1	Cache as Processor Source.....	13-25
13-2	Cache as Processor Sink.....	13-25
20-1	Configuring Record and Playback in an EPN.....	20-1
23-1	Foreign Stage Dependency Graph.....	23-11
23-2	Preferences Dialog: Application Library Path.....	23-15
23-3	Select Path Variable Dialog.....	23-16
23-4	New Variable Dialog.....	23-16
23-5	Select Path Variable: With Variable.....	23-17
23-6	Variable Extension Dialog.....	23-17
23-7	Preferences Dialog: Application Library Path With Path Variable.....	23-18
23-8	Oracle Event Processing IDE for Eclipse lib Directory.....	23-21
23-9	New Java Class Dialog.....	23-22
23-10	Manifest Editor: Overview Tab.....	23-24
23-11	Manifest Editor: Runtime Tab.....	23-25
23-12	JAR Selection Dialog.....	23-25
23-13	Manifest Editor: Dependencies Tab.....	23-26
23-14	Package Selection Dialog.....	23-27
23-15	Manifest Editor.....	23-28
24-1	Oracle Event Processing High Availability: Primary and Secondary Servers.....	24-2
24-2	Oracle Event Processing High Availability Lifecycle State Diagram.....	24-2
24-3	Secondary Failure.....	24-3
24-4	Primary Failure and Failover.....	24-3
24-5	High Availability Adapters in the EPN.....	24-5
24-6	High Availability and Scalability.....	24-8
24-7	Precise Recovery with JMS.....	24-11
24-8	Event Order.....	24-16
24-9	Simple Failover EPN.....	24-20
24-10	Simple Failover With Buffering EPN.....	24-23
24-11	Light-Weight Queue Trimming EPN.....	24-27
24-12	Precise Recovery With JMS EPN.....	24-35
25-1	Event Partitioner EPN.....	25-2
25-2	Oracle Event Processing ActiveActiveGroupBean Without High Availability.....	25-5
25-3	Oracle Event Processing ActiveActiveGroupBean With High Availability.....	25-6
25-4	EventPartitioner EPN.....	25-7

25-5	New Java Class Dialog .....	25-11
25-6	Precise Recovery With JMS EPN .....	25-17
25-7	Oracle Event Processing ActiveActiveGroupBean With High Availability.....	25-19
D-1	EPN With Oracle CQL Processor and Down-Stream Channel .....	D-76

---

---

# Preface

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

Oracle Event Processing (formerly 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 Event Processing 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 Event Processing supports ultra-high throughput and microsecond latency using JRockit Real Time and provides Oracle Event Processing Visualizer and Oracle Event Processing 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 Event Processing applications.

## Documentation Accessibility

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

### Access to Oracle Support

Oracle customers that have purchased support have access to electronic support through My Oracle Support. For information, visit <http://www.oracle.com/pls/topic/lookup?ctx=acc&id=info> or visit <http://www.oracle.com/pls/topic/lookup?ctx=acc&id=trs> if you are hearing impaired.

## Related Documents

For more information, see the following:

- *Oracle Fusion Middleware Getting Started Guide for Oracle Event Processing*
- *Oracle Fusion Middleware Administrator's Guide for Oracle Event Processing*
- *Oracle Fusion Middleware Visualizer User's Guide for Oracle Event Processing*

- *Oracle Fusion Middleware Java API Reference for Oracle Event Processing*
- *Oracle Fusion Middleware CQL Language Reference for Oracle Event Processing*
- *Oracle Fusion Middleware EPL Language Reference for Oracle Event Processing*
- *Oracle Database SQL Language Reference at*  
[http://download.oracle.com/docs/cd/B28359\\_01/server.111/b28286/toc.htm](http://download.oracle.com/docs/cd/B28359_01/server.111/b28286/toc.htm)
- 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 Processing Forum:  
<http://forums.oracle.com/forums/forum.jspa?forumID=820>
- Oracle Event Processing Samples:  
<http://www.oracle.com/technologies/soa/complex-event-processing.html>

## Conventions

The following text conventions are used in this document:

<b>Convention</b>	<b>Meaning</b>
<b>boldface</b>	Boldface type indicates graphical user interface elements associated with an action, or terms defined in text or the glossary.
<i>italic</i>	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.

---

---

## What's New in This Guide

For this release, this guide has been updated in several ways. The following table lists the sections that have been added or changed. If a feature was not available in the first release of 11.1.1.7.x, the last columns denote which documentation release contains the update.

For a list of known issues (release notes), see the "Known Issues for Oracle SOA Products and Oracle AIA Foundation Pack" at

<http://www.oracle.com/technetwork/middleware/docs/soa-aiAFP-knownissuesindex-364630.html>.

Sections	Changes Made	February 2013
<b>Entire Guide</b>	Product renamed to Oracle Event Processing	X
<b>Chapter 1 Overview of Creating Oracle Event Processing Applications</b>		
<a href="#">Section, "Key Concepts Underlying Oracle Event Processing Applications"</a>	Section added to describe key concepts in the Oracle Event Processing programming model.	X
<a href="#">Section, "How an Oracle Event Processing Application Works"</a>	Section added to provide an overview of how the components and technologies of an application work together.	X
<a href="#">Section, "Overview of Events, Streams and Relations"</a>	Section added to introduce events, streams and relations.	X
<b>Chapter 2 Oracle Event Processing Samples</b>		
<a href="#">Chapter 2, "Oracle Event Processing Samples"</a>	Chapter moved from <i>Oracle Fusion Middleware Getting Started Guide for Oracle Event Processing</i>	X
<a href="#">Section, "Signal Generation Example"</a>	Corrected sample description.	X
<b>Chapter 3 Getting Started with Developing Oracle Event Processing Applications</b>		
<a href="#">Chapter 3, "Getting Started with Developing Oracle Event Processing Applications"</a>	Chapter created from content formerly in an overview chapter.	X
<b>Chapter 8 Walkthrough: Assembling a Simple Application</b>		

<b>Sections</b>	<b>Changes Made</b>	<b>February 2013</b>
<a href="#">Chapter 8, "Walkthrough: Assembling a Simple Application"</a>	Chapter added. This is a developer's step-by-step introduction to building an Oracle Event Processing application.	X
<b>Chapter 9 Defining and Using Event Types</b>		
<a href="#">Chapter 9, "Defining and Using Event Types"</a>	Chapter updated with introductory information. Content on creating event types from non-JavaBean classes removed.	X
<b>Chapter 9 Defining and Using Event Types</b>		
<a href="#">Section, "Handling Faults in Channels"</a>	Added a summary on how you can handle Oracle CQL faults in channels by writing a fault handling class.	X
<b>Chapter 11 Integrating the Java Message Service</b>		
<a href="#">Section, "JMS Inbound Adapter Component Configuration"</a>	Updated the <code>session-ack-mode-name</code> <code>AUTO_ACKNOWLEDGE</code> description.	X
<b>Chapter 13 Integrating a Cache</b>		
<a href="#">Chapter 13, "Integrating a Cache"</a>	Chapter updated with introductory information and restructured for clarity.	X
<b>Chapter 16 Handling Events with Java</b>		
<a href="#">Chapter 16, "Handling Events with Java"</a>	Chapter added. Chapter presents information about implementing event sources and sinks in event beans and Spring beans.	X
<b>Chapter 17 Querying an Event Stream with Oracle CQL</b>		
<a href="#">Section, "Handling Faults"</a>	Section added to describe how to handle faults that occur in CQL code.	X
<b>Chapter 22 Testing Applications with Event Inspector</b>		
<a href="#">Chapter, "Injecting Events"</a> and <a href="#">Chapter, "Tracing Events"</a>	Sections updated with information describing how to configure event injection and tracing in component configuration files.	X
<b>Appendix D Schema Reference: Component Configuration <code>wlevs_application_config.xsd</code></b>		
<a href="#">Section, "heartbeat"</a>	Added content describing the <code>heartbeat</code> child element, which specifies a new heartbeat timeout for a system-timestamped channel.	X

# Part I

---

## Getting Started with Creating Oracle Event Processing Applications

Part I contains the following chapters:

- [Chapter 1, "Overview of Creating Oracle Event Processing Applications"](#)
- [Chapter 2, "Oracle Event Processing Samples"](#)
- [Chapter 3, "Getting Started with Developing Oracle Event Processing Applications"](#)





---

---

# Overview of Creating Oracle Event Processing Applications

This chapter introduces the tools, technologies, and processes through which you can build Oracle Event Processing applications, including the programming model, how applications work, and key concepts and technologies.

This chapter includes the following sections:

- [Oracle Event Processing Application Programming Model](#)
- [How an Oracle Event Processing Application Works](#)
- [Overview of Events, Streams and Relations](#)
- [Overview of Application Configuration](#)
- [Oracle Event Processing APIs](#)
- [Packaging an Application](#)
- [Oracle Event Processing Application Lifecycle](#)

## Oracle Event Processing Application Programming Model

This section provides an overview of the concepts, technologies, and tools that are part of building Oracle Event Processing applications.

An Oracle Event Processing application receives and processes data streaming from an event source. That data might be coming from one of a wide variety of places, including a monitoring device, a financial services company, or a motor vehicle. Using the data, the application might identify and respond to patterns, look for extraordinary events and alert other applications, or do some other work that requires immediate action based on quickly changing data.

When developing an Oracle Event Processing application, you assemble a network of components that each have a role in processing the data. This event processing network is essentially linear, with events passing through it from one end to the other. Along the way, components execute queries in a language specifically designed for streaming data, execute logic in Java, and create connections with other external components.

[Section , "Key Concepts Underlying Oracle Event Processing Applications"](#)

[Section , "Component Roles in an Event Processing Network"](#)

[Section , "Tools and Supporting Technologies for Developing Applications"](#)

## Key Concepts Underlying Oracle Event Processing Applications

Applications you build with Oracle Event Processing are based on concepts and technologies that are a mix of the familiar and (if you're new to event-based applications) unfamiliar. The following list briefly describes the key concepts and technologies that underlie how event processing networks work.

- **Applications leverage the database programming model.** Some of the programming model in Oracle Event Processing applications is conceptually an extension of what you find in database programming. Events are similar to database rows in that they are tuples against which you can execute queries (with a language that is an extension of SQL). In other words, if you know relational databases, much of this will seem familiar.

For a closer look at the similarity between database programming and Oracle Event Processing applications, see [Section , "Overview of Events, Streams and Relations"](#).

- **Stages represent discrete functional roles.** The staged structure of an event processing network (EPN) provides a means for you to execute different kinds of logic against events flowing through the network. This includes query logic with Oracle Continuous Query Language (Oracle CQL) as well as logic in Java. It also provides a way to capture multiple processing paths with a network that branches into multiple downstream directions based on discoveries your code makes.

For a list of roles stages play in an EPN, see [Section , "Component Roles in an Event Processing Network"](#). For a closer look at the pieces of an EPN, see [Section , "How an Oracle Event Processing Application Works"](#).

- **Stages transmit events through an EPN by acting as event sinks and event sources.** The stages in an EPN are able to receive events (as event sinks) and/or send events (as event sources). This includes stage components that come with Oracle Event Processing as well as components you build, such as your own adapters and beans.

For more about implementing your own event sinks and sources, see [Chapter 16, "Handling Events with Java"](#)

- **Events are handled either as streams or relations.** The concept of a stream, unique to streaming event-based applications, captures the fact that events arrive at your application sequentially by timestamp. Contrast this with rows in a database, where the table rows have no inherent relationship to one another aside from schema. However, many queries of events in stream result in a relation, in which events could be related in a way other than their relative sequence in time (similar to database query results).

For more on streams and relations, see [Section , "Overview of Events, Streams and Relations"](#).

## Component Roles in an Event Processing Network

The core of Oracle Event Processing applications you build is an event processing network (EPN). You build an EPN by connecting components (also known as stages) that each have a role in processing events that pass through the network. When developing an Oracle Event Processing application, you identify which kinds of components will be needed. As you add components to the EPN, you configure each as well as their connections with one another. As you use the IDE to build the EPN, you arrange and connect components in a roughly linear shape in which events will enter from the left end, move through the EPN and exit or terminate at the right end.

For a high-level overview of an EPN using an application example, including fuller descriptions of the technologies involved, see [Section , "How an Oracle Event Processing Application Works"](#).

The EPN components you use provide ways to:

- **Exchange event data with external sources.** Through adapters and other stages, you can connect external components to the EPN of your application to add ways for data, including event data, to pass into or out of the EPN.

These external components include those in the following list (you can also build your own).

- Relational databases. You can access a database table from within Oracle CQL code, querying the database as you would with SQL.
- Caches. By adding a cache stage to an EPN, you can exchange data with the cache.
- Java Message Service (JMS). With the JMS adapter, you can exchange messages with a JMS destination without writing the Java code typically needed for it.
- HTTP publish-subscribe server. The HTTP pub-sub adapter simplifies exchanging messages with an HTTP publish-subscribe server.
- **Model event data so that it can be handled by application code.** You implement or define event types that model event data so that application code can work with it. For more information, see [Chapter 9, "Defining and Using Event Types"](#).
- **Query and filter events.** The Oracle Continuous Query Language (Oracle CQL) is an extension of the SQL language through which you can query events as you would data in a database. Oracle CQL includes features specifically intended for querying streaming data. You add Oracle CQL code to an event processing network by adding a processor. For more information, see [Chapter 17, "Querying an Event Stream with Oracle CQL"](#).
- **Execute Java logic to handle events.** To an EPN you can add Java classes that receive and send events just as other EPN stages do. Logic in these classes can retrieve values from events, create new events, and more. For more information, see [Chapter 16, "Handling Events with Java"](#).

For IDE reference information on creating event processing networks, see [Section , "Using the EPN Editor"](#).

## Tools and Supporting Technologies for Developing Applications

Included with Oracle Event Processing is a set of tools and supporting technologies you can use to develop applications. These include tools for building and debugging applications, testing applications in a lightweight way, accessing underlying functionality with Java, and designing Oracle CQL queries.

The following lists some of these tools and technologies:

- The **Oracle Event Processing IDE for Eclipse** provides features specifically designed to make developing Oracle Event Processing easier, including an EPN Editor for graphically designing an event processing network. For more information, see [Chapter 4, "Overview of the Oracle Event Processing IDE for Eclipse"](#).

For a step-by-step introduction to using the IDE to build a simple application, see [Chapter 8, "Walkthrough: Assembling a Simple Application"](#).

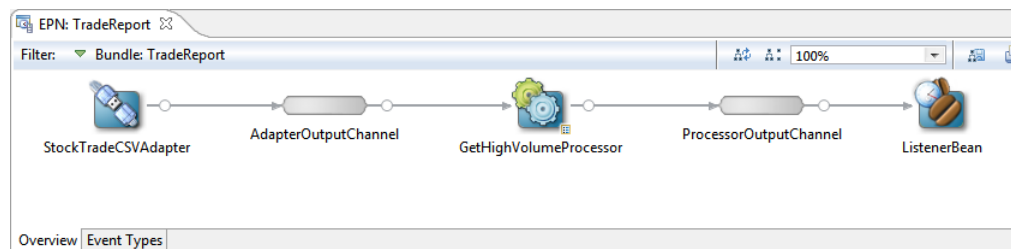
- By using the **load generator with the csvgen adapter**, you can more easily debug an application in the early stages of development. The load generator is a tool that reads data from a comma-separated text file and feeds the data to your EPN as event data. For more information, see [Chapter 21, "Testing Applications With the Load Generator and csvgen Adapter"](#).
- The **Oracle Event Processing Java API** includes classes for work such as implementing stages in the event processing network, extending included functionality, managing the server, and so on. For more information, see [Section , "Oracle Event Processing APIs"](#).
- **Oracle Event Processing Visualizer** is a web-based user interface through which you can design Oracle CQL queries and configure Oracle Event Processing applications on the server. For more information, see "Overview of Using Oracle Event Processing Visualizer" in the *Oracle Fusion Middleware Visualizer User's Guide for Oracle Event Processing*
- You connect and configure components through **standard XML configuration files**. Much of the work of writing these files is done for you when you use the IDE to assemble the event processing network. Yet there are many settings you might need to make by editing the files directly. For more on configuration files, see [Section , "Overview of Application Configuration"](#).
- You can **improve scalability and promote high availability** by using certain application design patterns, server resources, and configuration conventions. For more information, see [Chapter 25, "Developing Scalable Applications"](#) and [Chapter 24, "Developing Applications for High Availability"](#).

## How an Oracle Event Processing Application Works

As with many enterprise-scale applications, an Oracle Event Processing application is full of "connected-to" relationships. For example, an adapter might be connected to a processor, which might be connected to an event bean, which might be connected to an external data source, and so on. The connections aren't necessarily in that order, but you get the idea.

Events arrive from an outside source, then move through the application's event processing network (EPN). Along the way they might be filtered, queried, and otherwise processed as needed by EPN components that you put in place.

For example, take a look at a simple TradeReport application you can build using the topics described in [Chapter 8, "Walkthrough: Assembling a Simple Application"](#).



The following sections describe the role of each component in the application.

### **Event Information is Received in Its Raw Form**

In the TradeReport example, the event data source is simply a text file with rows of comma-separated values. To try things out, you can use such a file in combination with the load generator included with Oracle Event Processing.

An event data source is outside the application, yet connected through an adapter that knows how to retrieve its data. The event source could be something physically near (such as another application in the organization) or it could be quite far away (perhaps a temperature sensor in a server room in another city).

For information on event data and creating event types, see [Chapter 9, "Defining and Using Event Types"](#).

For step-by-step content on capturing event data in the TradeReport application, see [Section , "Create an Event Type to Carry Event Data"](#).

### **Adapters Connect External Components to the EPN**

In the TradeReport example, the event data source is connected to the event processing network through an adapter that knows how to receive event data sent from the CSV file. The adapter converts the incoming data into instances of an event type that the EPN can work with.

You can use an adapter to either receive incoming or send outgoing data. Adapters provided with Oracle Event Processing give you access to CSV files and also to systems such as the Java Message Service or an HTTP Publish-Subscribe server. You can also develop your own adapters for integrating systems that aren't supported by default.

When configuring an input adapter, you specify how the event data should be bound to an instance of an event type that you've defined in the EPN.

For more information, see the following:

[Chapter 11, "Integrating the Java Message Service"](#)

[Chapter 12, "Integrating an HTTP Publish-Subscribe Server"](#)

[Chapter 13, "Integrating a Cache"](#)

[Chapter 15, "Integrating an External Component Using a Custom Adapter"](#)

[Chapter 21, "Testing Applications With the Load Generator and csvgen Adapter"](#)

For step-by-step content on adding a csvgen adapter to receive CSV content in the TradeReport application, see [Section , "Add an Input Adapter to Receive Event Data"](#).

### **Event Types Provide Useful Structure for Event Data**

In the TradeReport application, the input adapter converts the incoming event data from a set of comma-separated values into property values of an event type instance that's defined as a Java class.

An event type provides a predictable structure for event data that other code in your application can use. Examples of that code include an Oracle CQL query that filters the events to discover those to act on, Java code in a bean that creates new kinds of events based on what your application received, and code to retrieve values and merge them with values from another data source.

Event types define properties that provide access to event data. Adapters receive incoming events from different event sources, such as the Java Messaging System (JMS) or financial market data feeds. You must define an Oracle Event Processing event type for these events before code is able to work with them.

You specify event types when you're configuring the EPN. With the structure of the raw event data in hand, you can define an event type that best suits your application's needs. The general best practice for defining an event type is to write a JavaBean class whose properties map to the event data that your application will use. You can also define event types as simple tuples and Java Map instances.

For more information, see [Chapter 9, "Defining and Using Event Types"](#)

For step-by-step content on defining an event type for the TradeReport application, see [Section , "Create an Event Type to Carry Event Data"](#).

### **Channels Transfer Events from Stage to Stage**

In the TradeReport application, the input adapter transmits events to an Oracle CQL processor by sending the events through a channel.

Channels connect stages in an event processing network. Between most components you will add a channel that listens for events coming from one stage and sends those events to another stage.

For more information on adding channels, see [Chapter 10, "Connecting EPN Stages Using Channels"](#).

For step-by-step content on adding a channel to the TradeReport application, see [Section , "Add a Channel to Convey Events"](#).

### **Processors Contain Query Code to Examine Events**

The TradeReport application includes a processor that contains a simple Oracle Continuous Query Language (Oracle CQL) query to filter the stream of received events down to only those that meet certain criteria. As events flow through the processor, the Oracle CQL code executes, acting as a filter to determine which events get passed to the next stage.

The logic of the application relies on the specific qualities of events flowing through it. In this application, a processor stage is a place to discover those qualities by using Oracle CQL code to look for occurrences of particular data or trends.

Because it is a great deal like the SQL language used to query more static, relational data sources, the Oracle CQL language is a powerful tool to discover information about data represented by events. For example, Oracle CQL contains a wide assortment of functions ranging from simple `count()` and `sum()` functions to sophisticated statistics functions.

Oracle CQL is like SQL, but also includes functionality intended specifically for writing queries that account for a characteristic typically not as important when querying static databases: the passage of time. Through these time-related features, you can, for example, write code that defines specific windows, such as from the present to the preceding 5 milliseconds. Your queries can execute to isolate this window as events pass through.

You can also use the code in Oracle CQL processor to collect and combine data from a variety of sources, including a cache and a relational database. Using a cache, for example, gives you a place to put frequently-retrieved data where performance will be faster.

The Oracle CQL engine is also extensible with cartridges that make additional functionality available to your Oracle CQL code.

For more information on Oracle CQL, see [Chapter 17, "Querying an Event Stream with Oracle CQL"](#).

For step-by-step information on defining an Oracle CQL query in the TradeReport application, see [Section , "Add an Oracle CQL Processor to Filter Events"](#).

### **Beans Provide a Place for Java Logic**

Once the TradeReport application has processed events with Oracle CQL, it passes the resulting events to an event bean that receives the events and prints to the console data contained in each event type instance.

In an EPN, a bean provides a place for you to execute Java code over events that are passing through. A bean can receive and sent events. For example, a bean might receive events of one type, then retrieve their data to perform some calculation or lookup using the data. The bean could then create new events from the newly generated data before passing the new events along to another stage.

You can write and configure beans as either Spring beans or event beans. Spring beans are managed by the Spring framework, and are a good choice if you want to integrate your bean to an existing Spring deployment. Event beans, on the other hand, use Oracle Event Processing conventions for configuring beans so that they're managed by the Oracle Event Processing server. With an event bean, for example, you get the support of Oracle Event Processing server features such as monitoring and event recording and playback (useful for debugging an application).

For more information on writing and using Java classes that handle events, see [Chapter 16, "Handling Events with Java"](#).

For step-by-step content on adding an event bean to the TradeReport application, see [Section , "Create a Listener to Receive and Report Events"](#).

### **Configuration Files Define an EPN and Its Components**

The EPN presence of and connections between TradeReport application stages is configured in an EPN assembly XML file. There is also a component configuration file which, though it isn't used in this case, could define runtime configuration for the components.

The EPN assembly file is what you are writing as you assemble an event processing network using the IDE (the EPN editor is essentially a user interface for designing EPN assembly files). What's in the EPN assembly file declares the stages and determines how they interact, including which direction events flow when moving from one stage to another. The EPN assembly file also sets default values for component settings, or values that you won't need to change at runtime.

With a component configuration file, on the other hand, you can specify configuration data that an administrator can later change while the application is running. The component configuration file is where Oracle CQL code is typically written (as configuration for a processor component).

For more information on configuration files, see [Section , "Overview of Application Configuration"](#).

### **Design and Configuration Conventions Scalability and High Availability**

While the simple TradeReport application doesn't demonstrate them, there are design patterns that you can use to ensure that your application remains available and scales well.

When ensuring that your application remains highly available, you integrate application design patterns, server resources, and configuration conventions so that your deployed application continues to fulfill its role even in the event of software or hardware failures.

A scalable Oracle Event Processing application incorporates design patterns with implementation and configuration conventions to ensure that the application functions well as the event load increases.

For more information, see the following:

[Chapter 24, "Developing Applications for High Availability"](#)

[Chapter 25, "Developing Scalable Applications"](#)

## Overview of Events, Streams and Relations

An Oracle Event Processing application handles events that arrive in a stream as raw event data, are converted to event type instances inside the application, and move from one application stage to another in an event processing network. Along the way, the events might be filtered with Oracle CQL queries, handled by Java code, stored in a cache, forwarded to other applications, and so on.

But what is an event? With the emphasis on the streaming aspect of event data, it can be easy to forget how much events are like rows in a database. In application terms, an event is a tuple, or set of values. Like a relational database row, an event has a schema in which each value has specific constraints, such as a particular data type. An event's schema defines its set of properties (where values will go) and their types.

Where events are unlike database rows is in the importance of time. In a stream of events, *when* an event arrives, including which event arrives before or after another event, can make all the difference. As a result, your application needs to be able to account for time and sequence.

For example, in an application that processes stock trades, event tuples made up of stock symbol, price, last price, percentage change, and volume would likely arrive one after the other in the order in which each trade was executed. Your application's logic might look for trades of one stock that occurred immediately after trades of another.

In other words, in an event processing application the sequence in which events occur in a stream is as important as the data types and values of each event property. As a result, conventions of the Oracle Event Processing programming model reflect the importance of time and sequence. Your code should be able to discover which events are related to one another based on certain criteria (such as a shared stock symbol). But it also needs to be able to discover sequence patterns (such as trades within fifteen seconds of one another). And it should be able to discover these things with very low latency *as the events arrive* at your application.

To account for both the sequential and relational aspects of event data, Oracle Event Processing uses the concepts of streams and relations.

- A *stream* is a potentially infinite sequence of events. Like rows in a database, the events are tuples, yet each has its own timestamp. In a stream, the events must be ordered by time, one after the other, so that timestamps do not decrease from one event to the next (although there might be events in a stream that have the same timestamp).
- In a *relation*, sequence might be unimportant (as with the results of a database query). Instead, events in a relation are typically related because they met certain criteria. For example, events in a relation might be the result of a query executed against a stream of stock trades, where the query was looking for trade volumes above a particular level.

Consider that stream of stock trade events. The events are arriving in sequence, each with its own timestamp (perhaps the time when the trade occurred). To isolate the



share price for trades that occurred within 5 seconds of one another, you query the stream (received from `StockTradeChannel`) with the following Oracle CQL code:

```
select price from StockTradeChannel [range 5 seconds]
```

Because it uses a window `-- [range 5 seconds] --` to isolate the events, this query's output is a relation. Though the events returned from the query have timestamps, they are unordered in the relation. Because the incoming events are in a stream, the query will execute continuously against every 5 seconds' worth of events as they pass into the query processor. As new events come along, those meeting the query terms are inserted into the relation, while those leaving (pushed out, actually) are deleted from the relation.

Why does this matter? A stream's integrity *as a stream* is important. Technically, a stream is a continuously moving -- well, streaming -- and ordered set of tuples. In a stream, every event can be said to be "inserted," having been put into the stream by its source, one after the other. When you get a subset of the stream with a query, you no longer have something that is ordered. And once you have the subset in hand, you might want to further isolate events by executing Oracle CQL code that queries the relations that result from queries of streams.

For this reason, examples in the Oracle CQL reference show the output of a query as events that are, as a result of the query and at that particular place in time, either inserted or deleted.

Before passing a relation along to the next stage in the EPN, you can convert it back into a stream by using an operator such as `IStream`.

For more information, see:

- [Section , "Channels Representing Streams and Relations"](#)
- "Streams and Relations" in the *Oracle Fusion Middleware CQL Language Reference for Oracle Event Processing*

## Overview of Application Configuration

You configure Oracle Event Processing applications through XML files that are based on standard schemas. When you install Oracle Event Processing, XSD files for these schemas are included in the directory `MIDDLEWARE_HOME\ocep_11.1\xsd`.

As you assemble an event processing network (EPN), even if you are using the IDE, you are creating an EPN assembly XML file. An entry for a stage in this file adds that stage to the EPN and defines its connections with other stages. For more on this file, see [Section , "Overview of EPN Assembly Files"](#).

Each component in your event processing network (adapter, processor, channel, or event bean) can have associated configuration XML. By providing this configuration, you provide a way for the component's configuration to be edited at runtime. (Only processors are *required* to have a configuration file.) Configuration XML for components can be grouped into a single component configuration file or divided among multiple files, depending on the needs of your development process. For more on component configuration files, see [Section , "Overview of Component Configuration Files"](#).

Other aspects of an Oracle Event Processing application might require their own configuration files. These include caching provided by Oracle Coherence. For information about other configuration, see documentation sections related to those technologies.

## Overview of EPN Assembly Files

When you are assembling an event processing network (EPN) using the IDE, you are defining the EPN in an assembly file. The EPN assembly file is an XML file whose shape is based on the Spring framework XML configuration file. The EPN assembly file schema extends the Spring configuration schema.

The `spring-wlevs-v11_1_1_6.xsd` schema file describes the structure of EPN assembly files. When you install Oracle Event Processing, XSD files such as this one are included in the directory `MIDDLEWARE_HOME\ocep_11.1\xsd.`

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 Event Processing 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_1_1_6.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 some Oracle Event Processing features, you specify some configuration in the EPN assembly file and some in the component configuration file.

For more information, see:

- [Section , "Creating EPN Assembly Files"](#)
- [Section , "Overview of Component Configuration Files"](#)

- [Appendix , "EPN Assembly Schema spring-wlevs-v11\\_1\\_1\\_6.xsd"](#)

### Nesting Stages in an EPN Assembly File

When you define a child stage within a parent stage in an EPN, 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="helloworldOutput" 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 , "Nested Stages"](#).

### Referencing Foreign Stages in an EPN Assembly File

You can refer to a stage that is in another Oracle Event Processing application. A stage from another application is considered a *foreign stage*. You do this by `id` attribute when you define both the source and target stage in the same application.

---

**Note:** You can't connect a processor stage to a channel that is a foreign stage.

---

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"/>
```

The following stages cannot be foreign stages:

- Cache

When creating Oracle Event Processing applications with foreign stages, you must consider foreign stage dependencies when assembling, deploying, and redeploying your application. For more information, see [Section , "Assembling Applications With Foreign Stages"](#).

## Overview of 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 Event Processing 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. When you install Oracle Event Processing, XSD files such as this one are included in the directory `MIDDLEWARE_HOME\ocep_11.1\xsd.`

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 Event Processing 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 , "Accessing Component and Server Configuration Using the ConfigurationPropertyPlaceholderConfigurer Class"](#)
- [Section , "Creating Component Configuration Files"](#)
- [Appendix , "Component Configuration Schema wlevs\\_application\\_config.xsd"](#)

### 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.

To use this feature, insert a `ConfigurationPropertyPlaceholderConfigurer` bean in the application context configuration file of your application bundle as [Example 1-5](#) shows.

#### **Example 1-5 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 Fusion Middleware Java API Reference for Oracle Event Processing*.

For more information on accessing property files, see [Section , "How to Add a Property File to an Oracle Event Processing Project"](#).

## Configuring Oracle Event Processing Resource Access

Because Oracle Event Processing applications are low latency high-performance event-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 Event Processing services, such as configuration management, to those provided by another Oracle product such as Oracle Kodo, to those provided by a third party.

By using Oracle Event Processing and standard Java annotations and deployment XML, you can configure the Oracle Event Processing Spring container to inject resources (such as data sources or persistence managers, and so on) into your Oracle Event Processing 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 Event Processing supports the following types of resource access:

- [Section , "Static Resource Injection"](#)
- [Section , "Dynamic Resource Injection"](#)
- [Section , "Dynamic Resource Lookup Using JNDI"](#)

See [Section , "Understanding Resource Name Resolution"](#) for information on resource name resolution.

See [Appendix I, "Oracle Event Processing Metadata Annotation Reference"](#) for complete details of all Oracle Event Processing annotations.

In the following sections, consider the example resource that [Example 1–6](#) shows. This is a data source resource named `StockDS` that you specify in the Oracle Event Processing server `config.xml` file.

### **Example 1–6 Sample Resource: Data Source StockDS**

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

## 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 , "Static Resource Names"](#)
- [Section , "Dynamic Resource Names"](#)

**Static Resource Names** When you configure static resource injection using static resource names, the resource name you use in the `@Resource` annotation or Oracle Event Processing 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 1-7](#) shows.

To override design time configuration at deploy time, you use Oracle Event Processing assembly file XML as [Example 1-8](#) shows.

In [Example 1-7](#) and [Example 1-8](#), the resource name `StockDs` exactly matches the name of the data source in the Oracle Event Processing server `config.xml` file as [Example 1-6](#) shows.

### **Example 1-7 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 1-8 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 1-9](#) shows. Similarly, in this case, the `wlevs:resource` element name attribute is not needed as [Example 1-10](#).

### **Example 1-9 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 1–10 Static Resource Injection Using Static Resource Names: XML**

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

**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 1–11](#) shows.

**Example 1–11 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 1–12](#) shows.

To override design time configuration at deploy time, you use Oracle Event Processing assembly file XML as [Example 1–13](#) shows.

**Example 1–12 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 1–13 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>
```

**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 1–14](#) shows.

**Example 1–14 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 runtime 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 , "com.bea.wlevs.configuration.Activate"](#)
- [Section , "com.bea.wlevs.configuration.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.

## Dynamic Resource Lookup Using JNDI

Oracle Event Processing supports the use of JNDI to look up resources dynamically as [Example 1–15](#).

### **Example 1–15 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 1–15](#), the JNDI name `StockDs` must exactly match the name of the data source in the Oracle Event Processing server `config.xml` file as [Example 1–6](#) shows.

---

**Note:** You must disable security when starting the Oracle Event Processing 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 Event Processing" in the *Oracle Fusion Middleware Administrator's Guide for Oracle Event Processing*.

---

## Understanding Resource Name Resolution

Oracle Event Processing server resolves resource names by examining the naming scopes that [Table 1–1](#) lists.

**Table 1–1 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

**Table 1–1 (Cont.) Resource Name Resolution**

Naming Scope	Contents	Resolution Behavior
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.

## Oracle Event Processing APIs

Oracle Event Processing 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.

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

For more information, see [Section , "Creating a Custom Adapter Factory"](#)

- 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.

For more information about the life cycle, see [Section , "Oracle Event Processing Application Lifecycle"](#).

- `InitializingBean`—Use if you require custom initialization after Oracle Event Processing has set all the properties of the component. Implement the `afterPropertiesSet` method.
- `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.
- `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 Event Processing 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.
- `ResumableBean`—Use if you want to perform some task, such as acquire or configure resources, before the component resumes work.
- `DisposableBean`—Use if you want to release resources when the application is undeployed. Implement the `destroy` method in your component code.

See also [Appendix I, "Oracle Event Processing Metadata Annotation Reference"](#) for additional lifecycle annotations.

- Event type instantiation interfaces—Use these interfaces for greater control over how event types are instantiated for use in an EPN.

For more information about event types, see [Chapter 1, "Overview of Creating Oracle Event Processing Applications"](#).

- `EventBuilder`—Use to control event type instantiation, such as to ensure that the properties of a configured event are correctly bound to the properties of an event type class, such as one you have implemented as a `JavaBean`.

For more information, see [Section , "Controlling Event Type Instantiation with an Event Type Builder Class"](#).

- `EventBuilder.Factory`—Factory for creating `EventBuilders`.
- Event source and sink interfaces—Use these to enable a class to receive and send events as part of the event processing network.

For more information on event sources and sinks, see [Section , "Handling Events with Sources and Sinks"](#) and

- `StreamSink` and `BatchStreamSink`—Components that want to receive events as an Oracle Event Processing stream must implement this interface. An Oracle Event Processing 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.

For more implementation information, see [Section , "Implementing an Event Sink"](#).

- `StreamSource`, `StreamSender` and `BatchStreamSender`—Components that send events modeling an Oracle Event Processing stream, such as adapters, must implement `StreamSource`. The interface has a `setEventSender` method for setting the `StreamSender` or `BatchStreamSender`, which actually send the event to the next component in the network.

For more implementation information, see [Section , "Implementing an Event Source"](#).

- `RelationSink` and `BatchRelationSink`—Components that want to receive events modeling an Oracle Event Processing relation must implement one of these interfaces. An Oracle Event Processing 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.

For more implementation information, see [Section , "Implementing an Event Sink"](#).

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

For more implementation information, see [Section , "Implementing an Event Source"](#).

For more information, see:

- For the full reference documentation for all classes and interfaces, see *Oracle Fusion Middleware Java API Reference for Oracle Event Processing*.
- For sample Java code that uses these APIs, see:
  - [Chapter 9, "Defining and Using Event Types"](#)

- [Chapter 15, "Integrating an External Component Using a Custom Adapter"](#)
- [Chapter 16, "Handling Events with Java"](#)
- [Chapter 2, "Oracle Event Processing Samples"](#)
- [Section , "Configuring Oracle Event Processing Resource Access"](#) for information on using Oracle Event Processing annotations and deployment XML to configure resource injection.

## Packaging an Application

After an application is assembled, it must be packaged so that it can be deployed into Oracle Event Processing. 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:

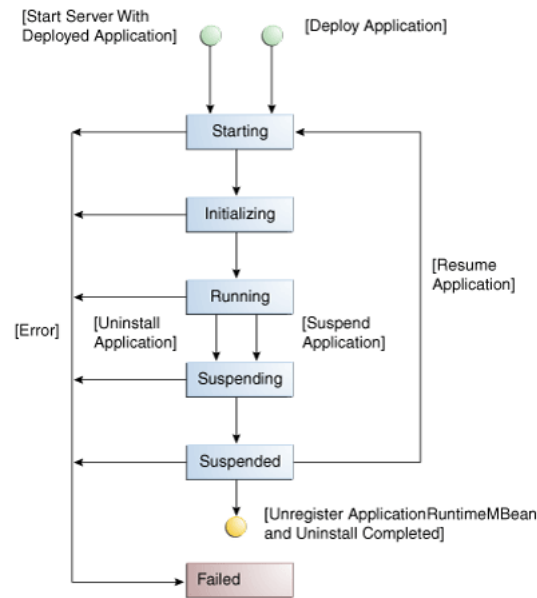
- The compiled application Java code of the business logic POJO.
- Component configuration files. Each processor is required to have a configuration file, although adapters and streams do not need to have a configuration file if the default configuration is adequate and you do not plan to monitor these 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 Event Processing so it can immediately start receiving incoming data.

For more information, see [Chapter 23, "Assembling and Deploying Oracle Event Processing Applications"](#).

## Oracle Event Processing Application Lifecycle

[Figure 1–1](#) shows a state diagram for the Oracle Event Processing 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 Event Processing; they are not OSGi bundle states.

**Figure 1–1 Oracle Event Processing Application Lifecycle State Diagram**


---

**Note:** For information on Oracle Event Processing server lifecycle, see "Oracle Event Processing Server Lifecycle" in the *Oracle Fusion Middleware Administrator's Guide for Oracle Event Processing*.

---

This section describes the lifecycle of an application deployed to the Oracle Event Processing server and the sequence of `com.bea.wlevs.ede.api` API callbacks.

This information explains how Oracle Event Processing manages an application's lifecycle so that you can better use the lifecycle APIs in your application.

For a description of these lifecycle APIs (such as `RunnableBean` and `SuspendableBean`), see:

- [Section, "Oracle Event Processing APIs"](#)
- [Appendix I, "Oracle Event Processing Metadata Annotation Reference"](#)
- *Oracle Fusion Middleware Java API Reference for Oracle Event Processing*

The lifecycle description is broken down into actions that a user performs, including those described in the following sections.

### Installing an application or starting the server with application already deployed

Oracle Event Processing performs the following actions:

1. Oracle Event Processing installs the application as an OSGI bundle. OSGI resolves the imports and exports, and publishes the service.
2. Oracle Event Processing creates beans (for both standard Spring beans and those that correspond to the Oracle Event Processing tags in the EPN assembly file). For each bean, Oracle Event Processing:
  - 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 the `InitializingBean.afterPropertiesSet` method.
- Calls configuration callbacks (`@Prepare,@Activate`) on Spring beans as well as factory-created stages.

For more information, see [Section , "Configuring Oracle Event Processing Resource Access"](#).

3. Application state is now `INITIALIZING`.
4. Oracle Event Processing registers the MBeans.
5. Oracle Event Processing calls the `ActivatableBean.afterConfigurationActive` method on all `ActivatableBeans`.
6. Oracle Event Processing calls the `ResumableBean.beforeResume` method on all `ResumableBeans`.
7. For each bean that implements `RunnableBean`, Oracle Event Processing starts it running in a thread.
8. Application state is now `RUNNING`.

### **Suspending the application**

Oracle Event Processing performs the following actions:

1. Oracle Event Processing calls the `SuspendableBean.suspend` method on all `SuspendableBeans`.
2. Application state is now `SUSPENDED`.

### **Resuming the application**

Oracle Event Processing performs the following actions:

1. Oracle Event Processing calls the `ResumableBean.beforeResume` method on all `ResumableBeans`
2. For each bean that implements `RunnableBean`, Oracle Event Processing starts it running in a thread.
3. Application state is now `RUNNING`.

### **Uninstalling application**

Oracle Event Processing performs the following actions:

1. Oracle Event Processing calls the `SuspendableBean.suspend` method on all `SuspendableBeans`.
2. Oracle Event Processing unregisters MBeans.
3. Oracle Event Processing calls the `DisposableBean.dispose` method on all `DisposableBeans`.
4. Oracle Event Processing uninstalls application bundle from OSGI.

### **Updating the application**

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

See those user actions in this list.

**Calling methods of stream and relation sources and sinks**

You may not call a method on a stream or relation source or sink from a lifecycle callback because components may not be ready to receive events until after these phases of the application lifecycle complete.

For example, you may not call `StreamSender` method `sendInsertEvent` from a lifecycle callback such as `afterConfigurationActive` or `beforeResume`.

You can call a method on a stream or relation source or sink from the `run` method of beans that implement `RunnableBean`.

For more information, see the description of installing an application. Also see [Section , "Handling Events with Sources and Sinks"](#).





---

---

## Oracle Event Processing Samples

This chapter introduces sample code provided with Oracle Event Processing, describing how to set up and use code ranging from simple "Hello World" to applications of Oracle Continuous Query Language (Oracle CQL), as well as for spatial and industry-focused scenarios.

This chapter includes the following sections:

- [Overview of the Samples Provided in the Distribution Kit](#)
- [Installing the Default ocep\\_domain and Samples](#)
- [Using Oracle Event Processing Visualizer With the Samples](#)
- [Increasing the Performance of the Samples](#)
- [HelloWorld Example](#)
- [Oracle Continuous Query Language \(Oracle CQL\) Example](#)
- [Oracle Spatial Example](#)
- [Foreign Exchange \(FX\) Example](#)
- [Signal Generation Example](#)
- [Event Record and Playback Example](#)

### Overview of the Samples Provided in the Distribution Kit

Oracle Event Processing includes the following samples:

- **HelloWorld:** a basic skeleton of a typical Oracle Event Processing application.
- **Oracle Continuous Query Language (CQL):** an example that shows how to use the Oracle Event Processing Visualizer Query Wizard to construct various Oracle CQL queries to process event streams.
- **Oracle Spatial:** an example that shows how to use Oracle Spatial with Oracle CQL queries to process a stream of Global Positioning System (GPS) events to track the GPS location of buses and generate alerts when a bus arrives at its pre-determined bus stop positions.
- **Foreign Exchange (FX):** a complete example that includes multiple components.
- **Signal Generation:** an example that simulates market trading and trend detection.
- **Event record and playback:** an example that shows how to configure event record and playback using a persistent event store.

These samples are provided in two forms, as follows:

- [Section , "Ready-to-Run Samples"](#)
- [Section , "Sample Source"](#)

The samples use Ant as their development tool; for details about Ant and installing it on your computer, see <http://ant.apache.org/>.

## Ready-to-Run Samples

Out-of-the-box sample domains pre-configured to deploy an assembled application; each sample has its own domain for simplicity. Each domain is a standalone server domain; the server files are located in the `defaultserver` subdirectory of the domain directory. To deploy the application you simply start the default server in the domain.

- The sample HelloWorld domain is located in `\MIDDLEWARE_HOME\occep_11.1\samples\domains\helloworld_domain`, where `MIDDLEWARE_HOME` refers to the Middleware directory you specified when you installed Oracle Event Processing, such as `d:\Oracle\Middleware`.  
[See Section , "Running the HelloWorld Example from the helloworld Domain"](#) for details.
- The sample CQL domain is located in `MIDDLEWARE_HOME\occep_11.1\samples\domains\cql_domain`.  
[See Section , "Running the CQL Example"](#) for details.
- The sample Oracle Spatial domain is located in `MIDDLEWARE_HOME\occep_11.1\samples\domains\spatial_domain`.  
[See Section , "Running the Oracle Spatial Example"](#) for details.
- The sample Foreign Exchange domain is located in `MIDDLEWARE_HOME\occep_11.1\samples\domains\fx_domain`.  
[See Section , "Running the Foreign Exchange Example"](#) for details.
- The sample Signal Generation domain is located in `MIDDLEWARE_HOME\occep_11.1\samples\domains\signalgeneration_domain`.  
[See Section , "Running the Signal Generation Example"](#) for details.
- The sample Record and Playback domain is located in `MIDDLEWARE_HOME\occep_11.1\samples\domains\recplay_domain`.  
[See Section , "Running the Event Record/Playback Example"](#) for details.

## Sample Source

The Java and configuration XML source for each sample is provided in a separate source directory that describes a sample development environment.

- The HelloWorld source directory is located in `MIDDLEWARE_HOME\occep_11.1\samples\source\applications\helloworld`, where `MIDDLEWARE_HOME` refers to the Middleware directory you specified when you installed Oracle Event Processing, such as `d:\Oracle\Middleware`.  
[See Section , "Implementation of the HelloWorld Example"](#) for details.
- The CQL source directory is located in `MIDDLEWARE_HOME\occep_11.1\samples\source\applications\cql`.  
[See Section , "Implementation of the CQL Example"](#) for details.

- The Oracle Spatial source directory is located in `MIDDLEWARE_HOME\ocep_11.1.1\samples\source\applications\spatial`.  
See [Section , "Implementation of the Oracle Spatial Example"](#) for details.
- The Foreign Exchange source directory is located in `MIDDLEWARE_HOME\ocep_11.1\samples\source\applications\fx`.  
See [Section , "Implementation of the FX Example"](#) for details.
- The Signal Generation source directory is located in `MIDDLEWARE_HOME\ocep_11.1\samples\source\applications\signalgeneration`.  
See [Section , "Implementation of the Signal Generation Example"](#) for details.
- The Record and Playback source directory is located in `MIDDLEWARE_HOME\ocep_11.1\samples\source\applications\replay`.  
See [Section , "Implementation of the Record and Playback Example"](#) for details.

## Installing the Default ocep\_domain and Samples

To install all Oracle Event Processing components *including* the default `ocep_domain` domain (with default passwords) and the samples, you must chose the `Custom` option to also install the samples. The `Typical` option does *not* include the default `ocep_domain` and samples.

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

## Using Oracle Event Processing Visualizer With the Samples

The Oracle Event Processing Visualizer is a Web 2.0 application that consumes data from Oracle Event Processing, displays it in a useful and intuitive way to system administrators and operators, and, for specified tasks, accepts data that is then passed back to Oracle Event Processing so as to change it configuration.

Visualizer is itself an Oracle Event Processing application and is automatically deployed in each server instance. To use it with the samples, be sure you have started the server (instructions provided for each sample below) and then invoke the following URL in your browser:

```
http://host:9002/wlevs
```

where `host` refers to the name of the computer hosting Oracle Event Processing; if it is the same as the computer on which the browser is running you can use `localhost`.

Security is disabled for the HelloWorld application, so you can click Logon at the login screen without entering a username and password. For the FX and signal generation samples, however, security is enabled, so use the following to logon:

```
User Id: wlevs
Password: wlevs
```

For more information about Oracle Event Processing Visualizer, see *Oracle Fusion Middleware Visualizer User's Guide for Oracle Event Processing*.

## Increasing the Performance of the Samples

To increase the throughput and latency when running the samples, and Oracle Event Processing applications in general, Oracle recommends the following:

- Use the JRockit JDK included in Oracle JRockit Real Time and enable the deterministic garbage collector by passing the `-dgc` parameter to the command that starts the Oracle Event Processing instance for the appropriate domain:

```
prompt> startwlevs.cmd -dgc
```

By default the deterministic garbage collector is disabled for the samples.

For more information on Oracle JRockit Real Time, see

<http://www.oracle.com/technology/products/jrockit/jrrt/index.html>.

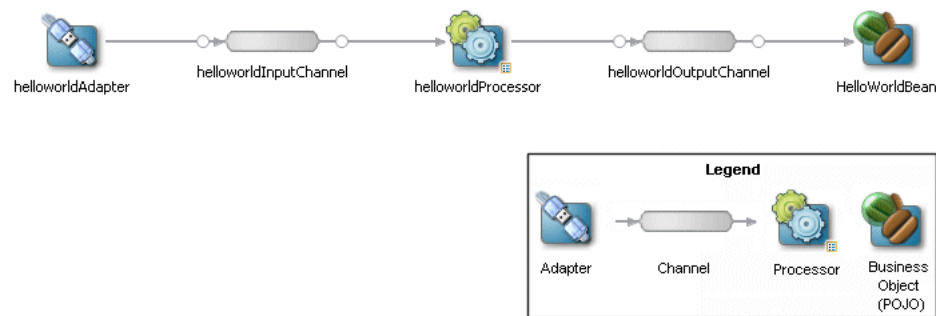
- When running Oracle Event Processing on a computer with a larger amount of memory, you should set the load generator and server heap sizes appropriately for the size of the computer. On computers with sufficient memory, Oracle recommend a heap size of 1 GB for the server and between 512MB - 1GB for the load generator.

## HelloWorld Example

The first example that shows how to create an Oracle Event Processing application is the ubiquitous HelloWorld.

Figure 2–1 shows the HelloWorld example Event Processing Network (EPN). The EPN contains the components that make up the application and how they fit together.

**Figure 2–1 The HelloWorld Example Event Processing Network**



The example includes the following components:

- `helloworldAdapter`—Component that generates *Hello World* messages every second. In a real-world scenario, this component would typically read a stream of data from a source, such as a data feed from a financial institution, and convert it into a stream of events that the event processor can understand. The HelloWorld application also includes a `HelloWorldAdapterFactory` that creates instances of `HelloWorldAdapter`.
- `helloworldInputChannel`—Component that streams the events generated by the adapter (in this case *Hello World* messages) to the event processor.
- `helloworldProcessor`—Component that simply forwards the messages from the `helloworldAdapter` component to the POJO that contains the business logic. In a real-world scenario, this component would typically execute additional and possibly much more processing of the events from the stream, such as selecting a

subset of events based on a property value, grouping events, and so on using Oracle CQL.

- `helloworldOutputChannel`—Component that streams the events processed by the event processor to the POJO that contains the user-defined business logic.
- `helloworldBean`—POJO component that simply prints out a message every time it receives a batch of messages from the processor via the output channel. In a real-world scenario, this component would contain the business logic of the application, such as running reports on the set of events from the processor, sending appropriate emails or alerts, and so on.

## Running the HelloWorld Example from the helloworld Domain

The HelloWorld application is pre-deployed to the `helloworld` domain. To run the application, you simply start an instance of Oracle Event Processing server.

### To run the HelloWorld example from the helloworld domain:

1. Open a command window and change to the default server directory of the `helloworld` domain directory, located in `MIDDLEWARE_HOME\ocep_11.1\samples\domains\helloworld_domain\defaultserver`, where `MIDDLEWARE_HOME` refers to the Middleware home directory you specified when you installed Oracle Event Processing, such as `d:\Oracle\Middleware`.

```
prompt> cd d:\Oracle\Middleware\ocep_11.1\samples\domains\helloworld_
domain\defaultserver
```

2. Ensure the environment is set correctly in the server startup script.

For more information, see [Chapter 3, "Getting Started with Developing Oracle Event Processing Applications"](#)

3. Start Oracle Event Processing by executing the appropriate script with the correct command line arguments:

#### a. On Windows:

- \* If you are using the JRockit JDK included in Oracle JRockit Real Time, enable the deterministic garbage collector by passing the `-dgc` parameter to the command:

```
prompt> startwlevs.cmd -dgc
```

- \* If you are not using the JRockit JDK included in Oracle JRockit Real Time:

```
prompt> startwlevs.cmd
```

#### b. On UNIX:

- \* If you are using the JRockit JDK included in Oracle JRockit Real Time, enable the deterministic garbage collector by passing the `-dgc` parameter to the command:

```
prompt> startwlevs.sh -dgc
```

- \* If you are not using the JRockit JDK included in Oracle JRockit Real Time:

```
prompt> startwlevs.sh
```

After server status messages scroll by, you should see the following message printed to the output about every second:

```
Message: HelloWorld - the current time is: 3:56:57 PM
```

This message indicates that the HelloWorld example is running correctly.

## Building and Deploying the HelloWorld Example from the Source Directory

The HelloWorld sample source directory contains the Java source, along with other required resources such as configuration XML files, that make up the HelloWorld application. The `build.xml` Ant file contains targets to build and deploy the application to the helloworld domain.

For more information, see [Section , "Description of the Ant Targets to Build Hello World"](#).

### To build and deploy the HelloWorld example from the source directory:

1. If the helloworld Oracle Event Processing instance is not already running, follow the procedure in [Section , "Running the HelloWorld Example from the helloworld Domain"](#) to start the server.

You must have a running server to successfully deploy the rebuilt application.

2. Open a new command window and change to the HelloWorld source directory, located in `MIDDLEWARE_HOME\ocep_11.1\samples\source\applications\helloworld` where `MIDDLEWARE_HOME` is the Middleware home directory you specified when you installed Oracle Event Processing.

For example:

```
prompt> cd d:\Oracle\Middleware\ocep_
11.1\samples\source\applications\helloworld
```

3. Set your development environment as described in [Section , "Setting Your Development Environment."](#)
  4. Execute the `all` Ant target to compile and create the application JAR file:
- ```
prompt> ant all
```
5. Execute the `deploy` Ant target to deploy the application JAR file to Oracle Event Processing:

```
prompt> ant -Daction=update deploy
```

---

---

**Caution:** This target overwrites the existing helloworld application JAR file in the domain directory.

---

---

You should see the following message printed to the output about every second:

```
Message: HelloWorld - the current time is: 3:56:57 PM
```

This message indicates that the HelloWorld example has been redeployed and is running correctly.

## Description of the Ant Targets to Build Hello World

The `build.xml` file, located in the top level of the HelloWorld source directory, contains the following targets to build and deploy the application:

- `clean`—This target removes the `dist` and output working directories under the current directory.
- `all`—This target cleans, compiles, and JARs up the application into a file called `com.bea.wlevs.example.helloworld_11.1.1.4_0.jar`, and places the generated JAR file into a `dist` directory below the current directory.
- `deploy`—This target deploys the JAR file to Oracle Event Processing using the Deployer utility.

For more information, see "Deployer Command-Line Reference" in the *Oracle Fusion Middleware Administrator's Guide for Oracle Event Processing*.

## Implementation of the HelloWorld Example

The implementation of the HelloWorld example generally follows [Section , "Creating an Oracle Event Processing Application"](#).

Refer to that section for a task-oriented procedure that describes the typical development process.

The HelloWorld example, because it is relatively simple, does not use all the components and configuration files described in the general procedure for creating an Oracle Event Processing application.

All the example files are located relative to the `MIDDLEWARE_HOME\ocp_11.1\samples\source\applications\helloworld` directory, where `MIDDLEWARE_HOME` refers to the Middleware directory you specified when you installed Oracle Event Processing `c:\Oracle\Middleware`. Oracle recommends that you use this example directory setup in your own environment, although it is obviously not required.

The files used by the HelloWorld example include:

- An EPN assembly file that describes each component in the application and how all the components are connected together. The EPN assembly file extends the standard Spring context file. The file also registers the event types used in the application. You are required to include this XML file in your Oracle Event Processing application.

In the example, the file is called `com.bea.wlevs.example.helloworld-context.xml` and is located in the `META-INF/spring` directory.

- Java source file for the `helloworldAdapter` component.

In the example, the file is called `HelloWorldAdapter.java` and is located in the `src/com/bean/wlevs/adapter/example/helloworld` directory.

For a detailed description of this file and how to program the adapter Java files in general, see [Chapter 15, "Integrating an External Component Using a Custom Adapter"](#).

- Java source file that describes the `HelloWorldEvent` event type.

In the example, the file is called `HelloWorldEvent.java` and is located in the `src/com/bean/wlevs/event/example/helloworld` directory.

For a detailed description of this file, as well as general information about programming event types, see [Chapter 9, "Defining and Using Event Types"](#).

- An XML file that configures the `helloworldProcessor` and `helloworldOutputChannel` components. An important part of this file is the set of EPL rules that select the set of events that the HelloWorld application processes.

You are required to include a processor configuration file in your Oracle Event Processing application, although the adapter and channel configuration is optional.

In the example, the file is called `config.xml` and is located in the `META-INF/wlevs` directory.

- A Java file that implements the `helloworldBean` component of the application, a POJO that contains the business logic.

In the example, the file is called `HelloWorldBean.java` and is located in the `src/com/bean/wlevs/example/helloworld` directory.

For a detailed description of this file, as well as general information about programming event sinks, see [Chapter 16, "Handling Events with Java"](#).

- A `MANIFEST.MF` file that describes the contents of the OSGi bundle that will be deployed to Oracle Event Processing.

In the example, the `MANIFEST.MF` file is located in the `META-INF` directory.

For more information about creating this file, as well as a description of creating the OSGi bundle that you deploy to Oracle Event Processing, see [Section , "Overview of Application Assembly and Deployment"](#).

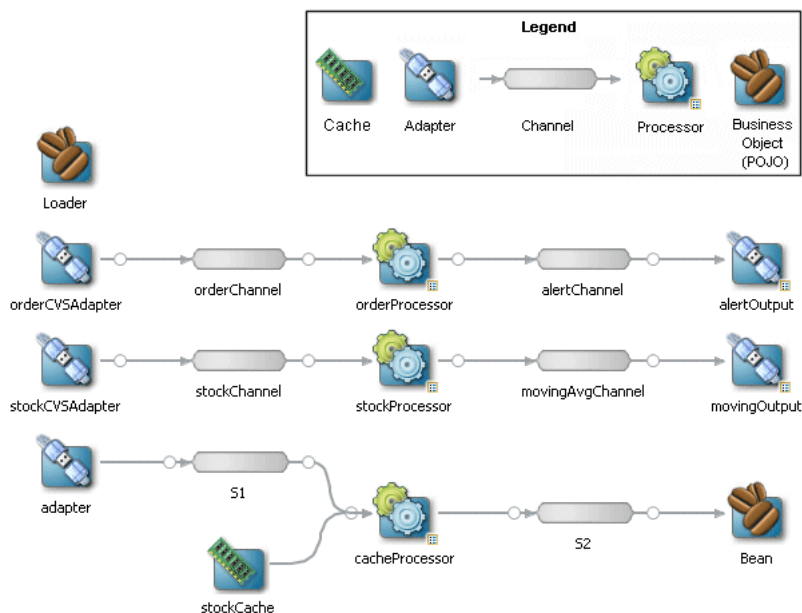
The HelloWorld example uses a `build.xml` Ant file to compile, assemble, and deploy the OSGi bundle; see [Section , "Building and Deploying the HelloWorld Example from the Source Directory"](#) for a description of this `build.xml` file if you also use Ant in your development environment.

## Oracle Continuous Query Language (Oracle CQL) Example

The Oracle CQL example shows how to use the Oracle Event Processing Visualizer Query Wizard to construct various types of Oracle CQL queries.

[Figure 2-2](#) shows the CQL example Event Processing Network (EPN). The EPN contains the components that make up the application and how they fit together.

**Figure 2-2 The CQL Example Event Processing Network**





The application contains three separate event paths in its EPN:

- **Missing events:** this event path consists of an adapter `orderCVSAdapter` connected to a channel `orderChannel`. The `orderChannel` is connected to processor `orderProcessor` which is connected to channel `alertChannel` which is connected to adapter `alertOutput`.

This event path is used to detect missing events in a customer order workflow.

For more information on how to construct the query that the `cqlProc` processor executes, see [Section , "Creating the Missing Event Query"](#).

- **Moving average:** this event path consists of channel `stockChannel` connected to processor `stockProcessor` which is connected to channel `movingAvgChannel` which is connected to adapter `movingOutput`.

This event path is used to compute a moving average on stock whose volume is greater than 1000.

For more information on how to construct the query that the `cqlProc` processor executes, see [Section , "Creating the Moving Average Query"](#).

- **Cache:** this event path consists of adapter `adapter` connected to channel `S1` connected to Oracle CQL processor `cacheProcessor` connected to channel `S2` connected to bean `Bean`. There is a cache `stockCache` also connected to the Oracle CQL processor `cacheProcessor`. There is also a bean `Loader`.

This event path is used to access information from a cache in an Oracle CQL query.

---

**Note:** For more information about the various components in the EPN, see the other samples in this book.

---

## Running the CQL Example

For optimal demonstration purposes, Oracle recommends that you run this example on a powerful computer, such as one with multiple CPUs or a 3 GHz dual-core Intel, with a minimum of 2 GB of RAM.

The CQL application is pre-deployed to the `cql_domain` domain. To run the application, you simply start an instance of Oracle Event Processing server.

### To run the CQL example:

1. Open a command window and change to the default server directory of the CQL domain directory, located in `MIDDLEWARE_HOME\ocep_11.1\samples\domains\cql_domain\defaultserver`, where `MIDDLEWARE_HOME` refers to the Middleware directory you specified when you installed Oracle Event Processing, such as `d:\Oracle\Middleware`.

```
prompt> cd d:\Oracle\Middleware\ocep_11.1\samples\domains\cql_
domain\defaultserver
```

2. Set your development environment, as described in [Chapter , "Setting Your Development Environment"](#)
3. Start Oracle Event Processing by executing the appropriate script with the correct command line arguments:
  - a. On Windows:

- \* If you are using the JRockit JDK included in Oracle JRockit Real Time, enable the deterministic garbage collector by passing the `-dgc` parameter to the command:

```
prompt> startwlevs.cmd -dgc
```

- \* If you are not using the JRockit JDK included in Oracle JRockit Real Time:

```
prompt> startwlevs.cmd
```

**b. On UNIX:**

- \* If you are using the JRockit JDK included in Oracle JRockit Real Time, enable the deterministic garbage collector by passing the `-dgc` parameter to the command:

```
prompt> startwlevs.sh -dgc
```

- \* If you are not using the JRockit JDK included in Oracle JRockit Real Time:

```
prompt> startwlevs.sh
```

The CQL application is now ready to receive data from the data feeds.

4. To simulate the data feed for the missing event query, open a new command window and set your environment as described in [Chapter 3, "Getting Started with Developing Oracle Event Processing Applications"](#)
5. Change to the `MIDDLEWARE_HOME\ocep_11.1\utils\load-generator` directory, where `MIDDLEWARE_HOME` refers to the Middleware directory you specified when you installed Oracle Event Processing, such as `d:\Oracle\Middleware`.
6. Run the load generator using the `orderData.prop` properties file:
  - a. On Windows:

```
prompt> runloadgen.cmd orderData.prop
```
  - b. On UNIX:

```
prompt> runloadgen.sh orderData.prop
```
7. To simulate the data feed for the moving average query, open a new command window and set your environment as described in [Chapter 3, "Getting Started with Developing Oracle Event Processing Applications"](#)
8. Change to the `MIDDLEWARE_HOME\ocep_11.1\utils\load-generator` directory, where `MIDDLEWARE_HOME` refers to the Middleware directory you specified when you installed Oracle Event Processing, such as `d:\Oracle\Middleware`.
9. Run the load generator using the `stockData.prop` properties file:
  - a. On Windows:

```
prompt> runloadgen.cmd stockData.prop
```
  - b. On UNIX:

```
prompt> runloadgen.sh stockData.prop
```
10. To simulate the data feed for the cache query, you only need to run the example.

The load data is generated by `Adaptor.java` and the cache data is generated by `Loader.java`. You can verify that data is flowing through by turning on statistics in the Oracle Event Processing Visualizer Query Plan.

## Building and Deploying the CQL Example

The CQL sample source directory contains the Java source, along with other required resources such as configuration XML files, that make up the CQL application. The `build.xml` Ant file contains targets to build and deploy the application to the `cql_domain` domain, as described in [Section , "Description of the Ant Targets to Build Hello World."](#)

### To build and deploy the CQL example from the source directory:

1. If the CQL Oracle Event Processing instance is not already running, follow the procedure in [Section , "Running the CQL Example"](#) to start the server.

You must have a running server to successfully deploy the rebuilt application.

2. Open a new command window and change to the CQL source directory, located in `MIDDLEWARE_HOME\ocep_11.1\samples\source\applications\cql`, where `MIDDLEWARE_HOME` refers to the Middleware directory you specified when you installed Oracle Event Processing, such as `d:\Oracle\Middleware`.

```
prompt> cd d:\Oracle\Middleware\ocep_11.1\samples\source\applications\cql
```

3. Set your development environment, as described in [Section , "Setting Your Development Environment"](#)

4. Execute the `all` Ant target to compile and create the application JAR file:

```
prompt> ant all
```

5. Execute the `deploy` Ant target to deploy the application JAR file to Oracle Event Processing:

```
prompt> ant -Dusername=wlevs -Dpassword=wlevs -Daction=update deploy
```

---

**Caution:** This target overwrites the existing CQL application JAR file in the domain directory.

---

6. If the load generators required by the CQL application are not running, start them as described in [Section , "Running the CQL Example."](#)

## Description of the Ant Targets to Build the CQL Example

The `build.xml` file, located in the top-level directory of the CQL source, contains the following targets to build and deploy the application:

- `clean`—This target removes the `dist` and output working directories under the current directory.
- `all`—This target cleans, compiles, and jars up the application into a file called `com.bea.wlevs.example.cql_11.1.1.4_0.jar`, and places the generated JAR file into a `dist` directory below the current directory.
- `deploy`—This target deploys the JAR file to Oracle Event Processing using the Deployer utility.

For more information, see "Deployer Command-Line Reference" in the *Oracle Fusion Middleware Administrator's Guide for Oracle Event Processing*.

## Implementation of the CQL Example

This section describes how to create the queries that the CQL example uses, including:

- [Section , "Creating the Missing Event Query"](#)
- [Section , "Creating the Moving Average Query"](#)

### Creating the Missing Event Query

This section describes how to use the Oracle Event Processing Visualizer Query Wizard to create the Oracle CQL pattern matching query that the `cqlProc` processor executes to detect missing events.

Consider a customer order workflow in which you have customer order workflow events flowing into the Oracle Event Processing system.

In a valid scenario, you see events in the order that [Table 2–1](#) lists:

**Table 2–1 Valid Order Workflow**

| Event Type | Description    |
|------------|----------------|
| C          | Customer order |
| A          | Approval       |
| S          | Shipment       |

However, it is an error if an order is shipped without an approval event as [Table 2–2](#) lists:

**Table 2–2 Invalid Order Workflow**

| Event Type | Description    |
|------------|----------------|
| C          | Customer order |
| S          | Shipment       |

You will create and test a query that detects the missing approval event and generates an alert event:

- ["To create the missing event query:"](#) on page 2-12
- ["To test the missing event query:"](#) on page 2-35

### To create the missing event query:

1. If the CQL Oracle Event Processing instance is not already running, follow the procedure in [Section , "Running the CQL Example"](#) to start the server.

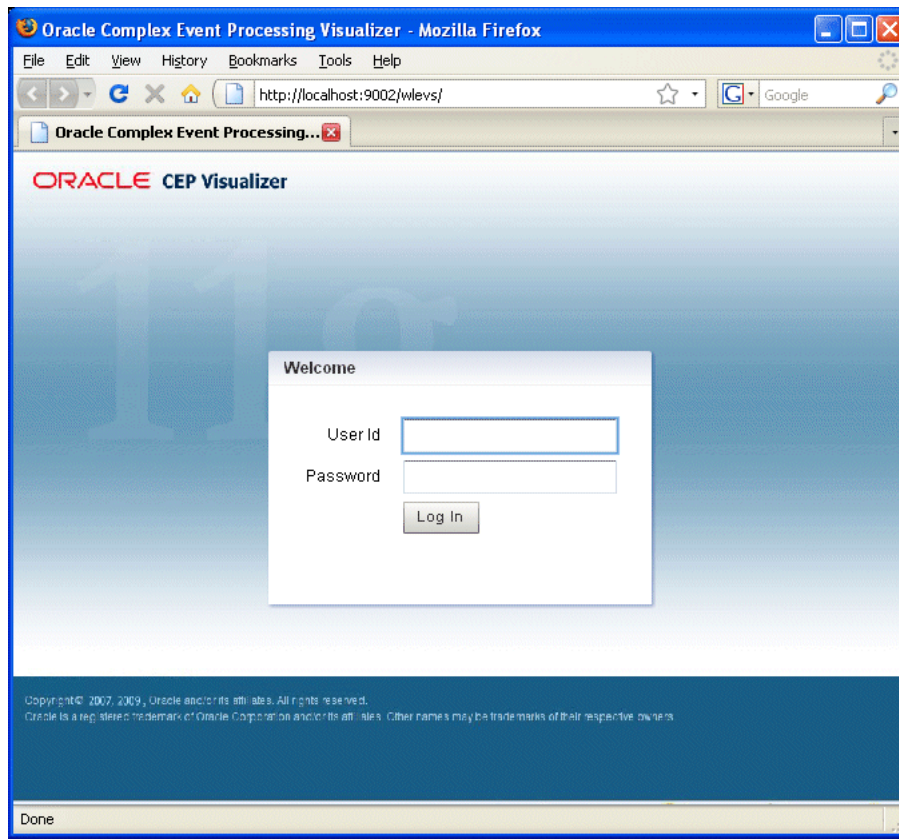
You must have a running server to use the Oracle Event Processing Visualizer.

2. Invoke the following URL in your browser:

```
http://host:port/wlevs
```

where *host* refers to the name of the computer on which Oracle Event Processing is running and *port* refers to the Jetty NetIO port configured for the server (default value 9002).

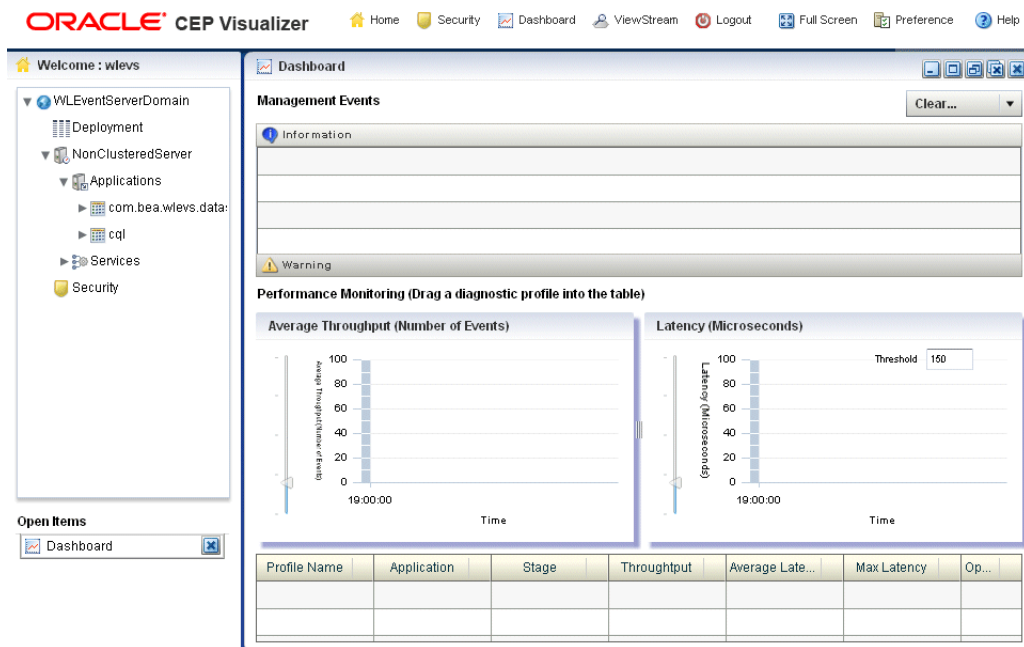
The Logon screen appears as [Figure 2–3](#) shows.

**Figure 2–3 Oracle Event Processing Visualizer Logon Screen**

3. In the Logon screen, enter the **User Id** `wlevs` and **Password** `wlevs`, and click **Log In**.

The Oracle Event Processing Visualizer dashboard appears as [Figure 2–4](#) shows.

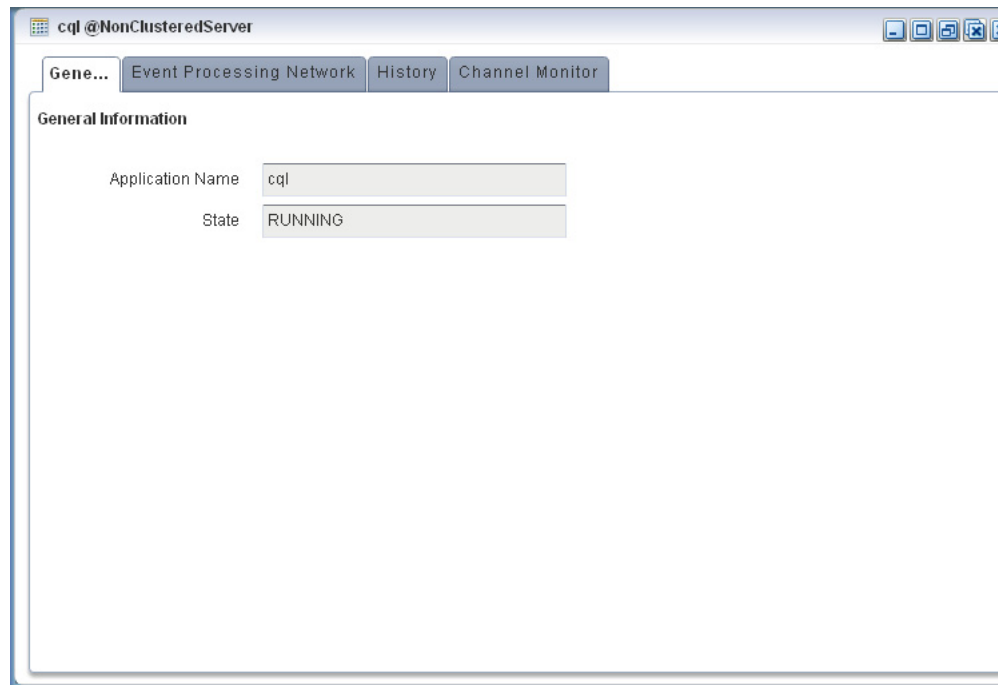
**Figure 2–4 Oracle Event Processing Visualizer Dashboard**



For more information about the Oracle Event Processing Visualizer user interface, see "Understanding the Oracle Event Processing Visualizer User Interface" in the *Oracle Fusion Middleware Visualizer User's Guide for Oracle Event Processing*.

4. In the right-hand pane, expand **WLEventServerDomain > NonClusteredServer > Applications**.
5. Select the **cql** node.

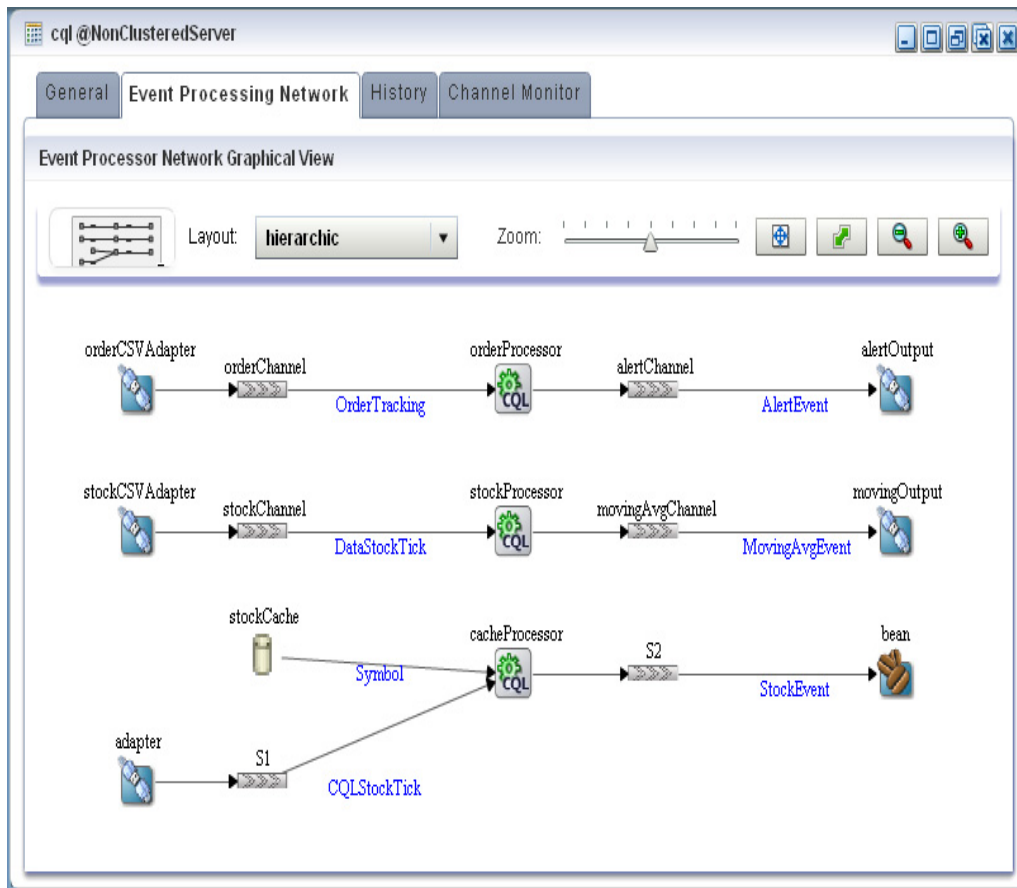
The CQL application screen appears as [Figure 2–5](#) shows.

**Figure 2–5 CQL Application Screen: General Tab**

**6. Select the **Event Processing Network** tab.**

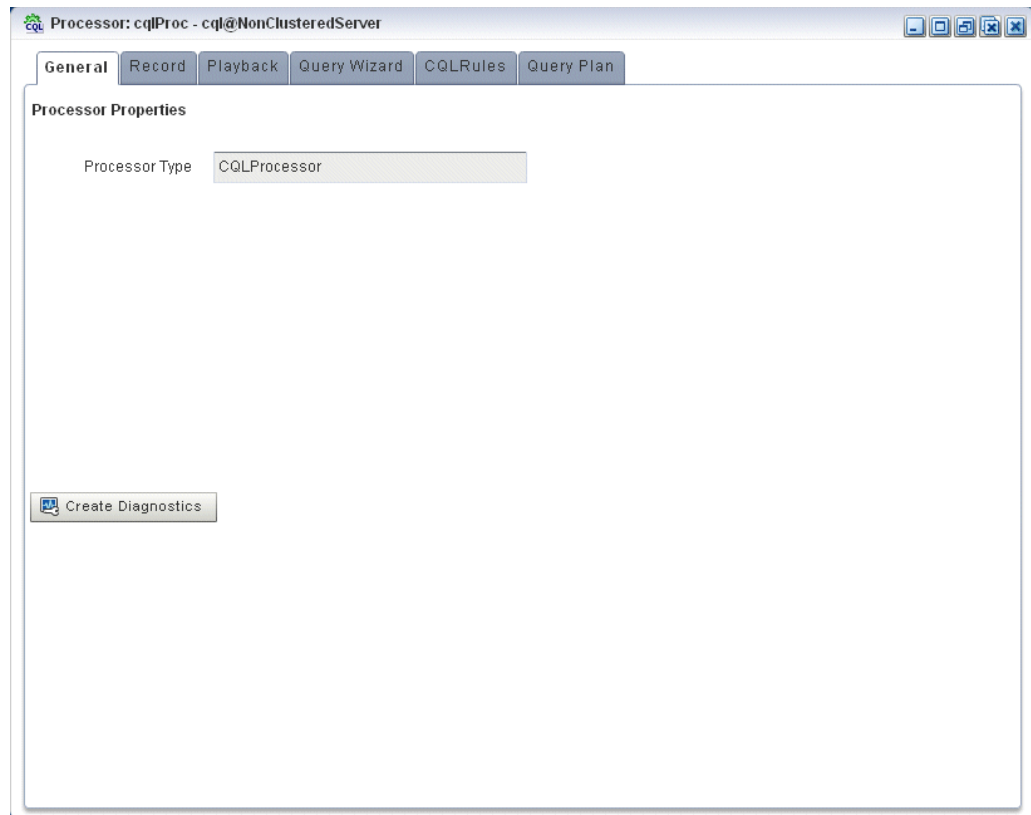
The Event Processing Network screen appears as [Figure 2–6](#) shows.

**Figure 2–6 CQL Application: Event Processing Network Tab**



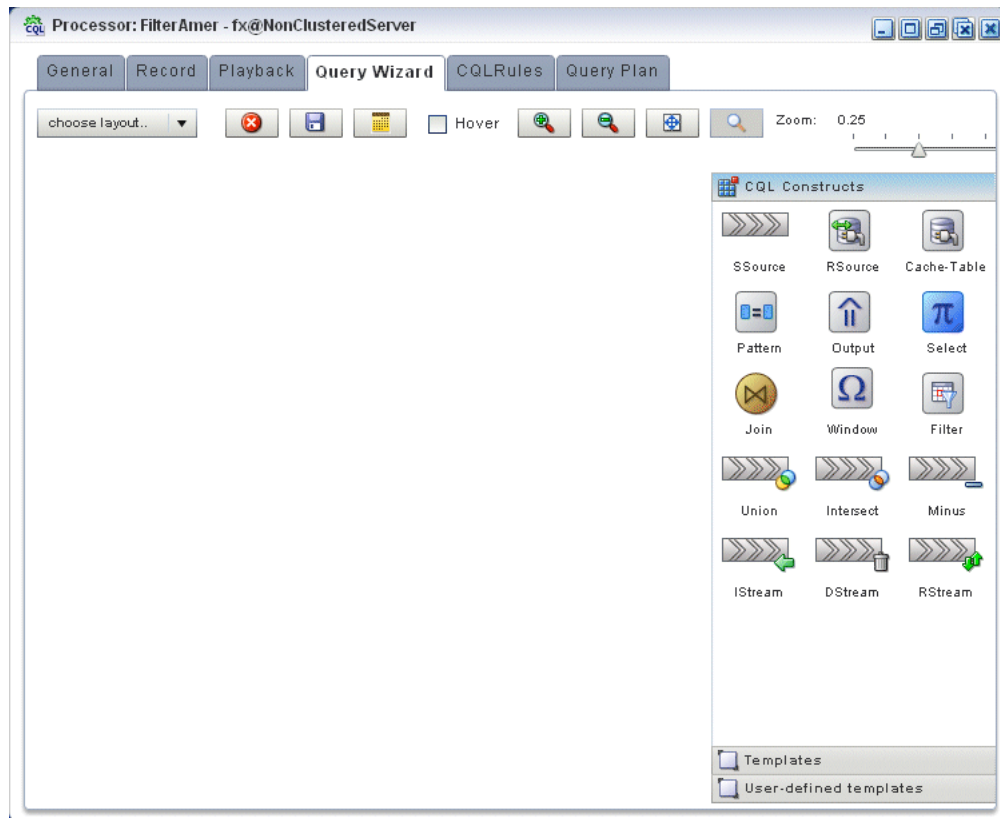
7. Double-click the **orderProcessor** Oracle CQL processor icon.  
The Oracle CQL processor screen appears as [Figure 2–7](#) shows.



**Figure 2–7 Oracle CQL Processor: General Tab**

8. Select the **Query Wizard** tab.

The Query Wizard screen appears as [Figure 2–8](#) shows.

**Figure 2–8 Oracle CQL Processor: Query Wizard Tab**

You can use the Oracle CQL Query Wizard to construct an Oracle CQL query from a template or from individual Oracle CQL constructs.

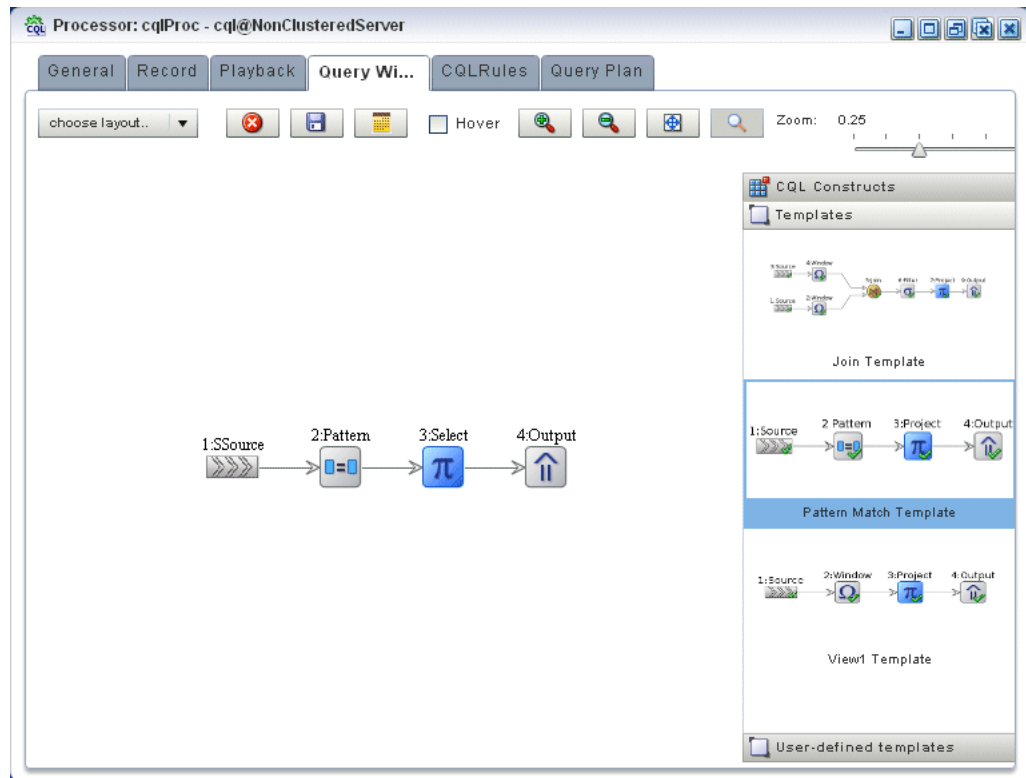
In this procedure, you are going to create an Oracle CQL query from a template.

For more information, see "Creating a Rule in an Oracle CQL Processor Using the Query Wizard" in the *Oracle Fusion Middleware Visualizer User's Guide for Oracle Event Processing*.

9. Click the Templates tab.

The Templates tab appears as [Figure 2–9](#) shows.

Figure 2–9 Template Tab



10. Click and drag the **Pattern Match Template** from the Templates palette and drop it anywhere in the Query Wizard canvas as shown in [Figure 2–9](#).
11. Double-click the **SSource** icon.  
The SSource configuration screen appears as [Figure 2–10](#) shows.

**Figure 2–10 SSource Configuration Dialog**

Stream [ID: 1]

Type  Stream  View

orderChannel AS

Source Properties

| Properties (4) |                  |
|----------------|------------------|
| amount         | java.lang.Long   |
| ts             | java.lang.String |
| eventType      | java.lang.String |
| orderid        | java.lang.String |

Generated CQL Statement

```
SELECT * FROM orderChannel
```

Help Validate Save Cancel

The source of your query will be the `orderChannel` stream.

12. Configure the SSource as follows:
  - Select **Stream** as the Type.
  - Select **orderChannel** from the **Select a source** pull-down menu.
13. Click **Save**.
14. Click **Save Query**.
15. Double-click the **Pattern** icon.

The Pattern configuration screen appears as [Figure 2–11](#) shows.

**Figure 2–11 Pattern Configuration Dialog: Pattern Tab**

Pattern Match [ ID : 2 ]

Pattern Define Subset Measure

Step 1 - Create Pattern

Pattern Expression: CustOrder NoApproval\*? Shipment  
(e.g. A B\*? C)

Duration: (e.g. 1 minute)

Partition By: orderid

Pattern Alias: Orders

All Matches

Generated Pattern Match Statement

```
SELECT * FROM orderChannel MATCH_RECOGNIZE ( PARTITION BY orderid PATTERN(CustOrder NoApproval*? Shipment)) AS Orders
```

Help Validate Save Cancel

Using the Pattern tab, you will define the pattern expression that matches when missed events occur. The expression is made in terms of named conditions that you will specify on the Define tab in a later step.

**16.** Enter the following expression in the Pattern Expression field:

CustOrder NoApproval\*? Shipment

This pattern uses the Oracle CQL pattern quantifiers that [Table 2–3](#) lists. Use the pattern quantifiers to specify the allowed range of pattern matches. The one-character pattern quantifiers are maximal or "greedy"; they will attempt to match the biggest quantity first. The two-character pattern quantifiers are minimal or "reluctant"; they will attempt to match the smallest quantity first.

**Table 2–3 MATCH\_RECOGNIZE Pattern Quantifiers**

| Maximal | Minimal | Description      |
|---------|---------|------------------|
| *       | *?      | 0 or more times  |
| +       | +?      | 1 or more times. |
| ?       | ??      | 0 or 1 time.     |

For more information, see:

- "PATTERN Condition" in the *Oracle Fusion Middleware CQL Language Reference for Oracle Event Processing*
- "MATCH\_RECOGNIZE Condition" in the *Oracle Fusion Middleware CQL Language Reference for Oracle Event Processing*

17. Select **orderid** from the **Partition By** pull-down menu and click the Plus Sign button to add this property to the `PARTITION BY` clause.

This ensures that Oracle Event Processing evaluates the missing event query on each order.

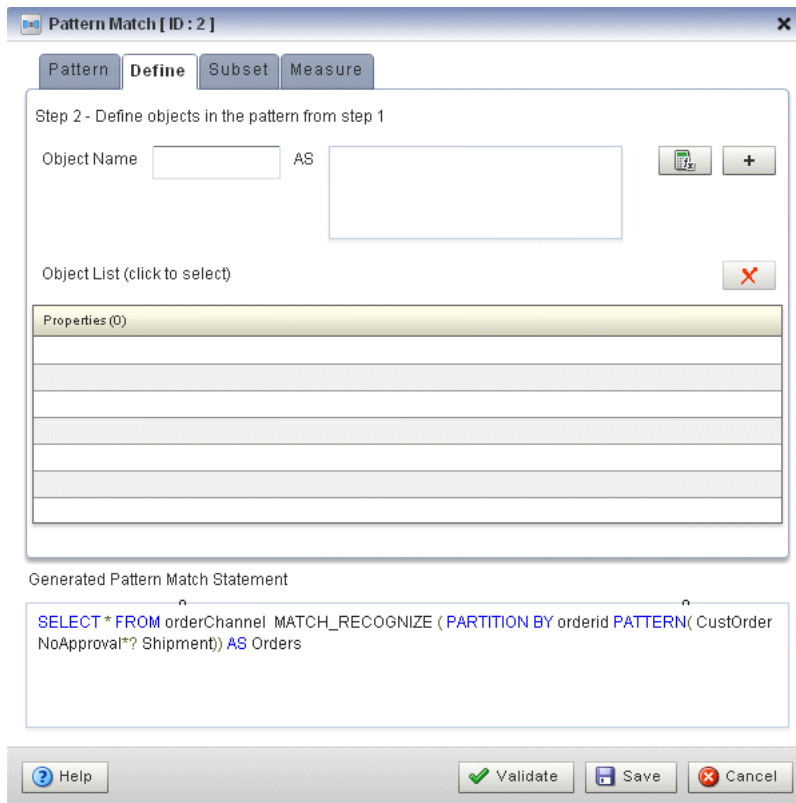
18. Enter **Orders** in the **Alias** field.

This assigns an alias (`Orders`) for the pattern to simplify its use later in the query.

19. Click the **Define** tab.

The Define tab appears as [Figure 2–12](#) shows.

**Figure 2–12 Pattern Configuration Dialog: Define Tab**



You will now define each of the conditions named in the pattern clause as [Table 2–4](#) lists:

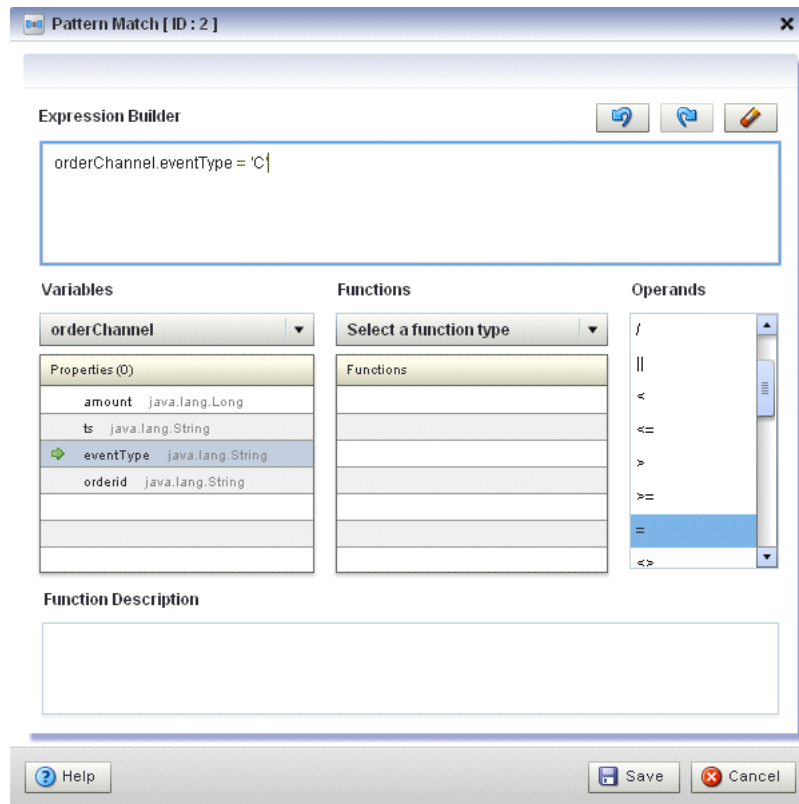
**Table 2–4 Condition Definitions**

| Condition Name | Definition                                      |
|----------------|-------------------------------------------------|
| CustOrder      | <code>orderChannel.eventType = 'C'</code>       |
| NoApproval     | <code>NOT (orderChannel.eventType = 'A')</code> |
| Shipment       | <code>orderChannel.eventType = 'C'</code>       |

20. Enter **CustOrder** in the **Object Name** field.
21. Click the Expression Builder button and configure the Expression Builder as follows (see [Figure 2–13](#)):
  - In the **Variables** list, double-click **eventType**.

- In the **Operands** list, double-click =.
- After the = operand, enter the value 'C'.

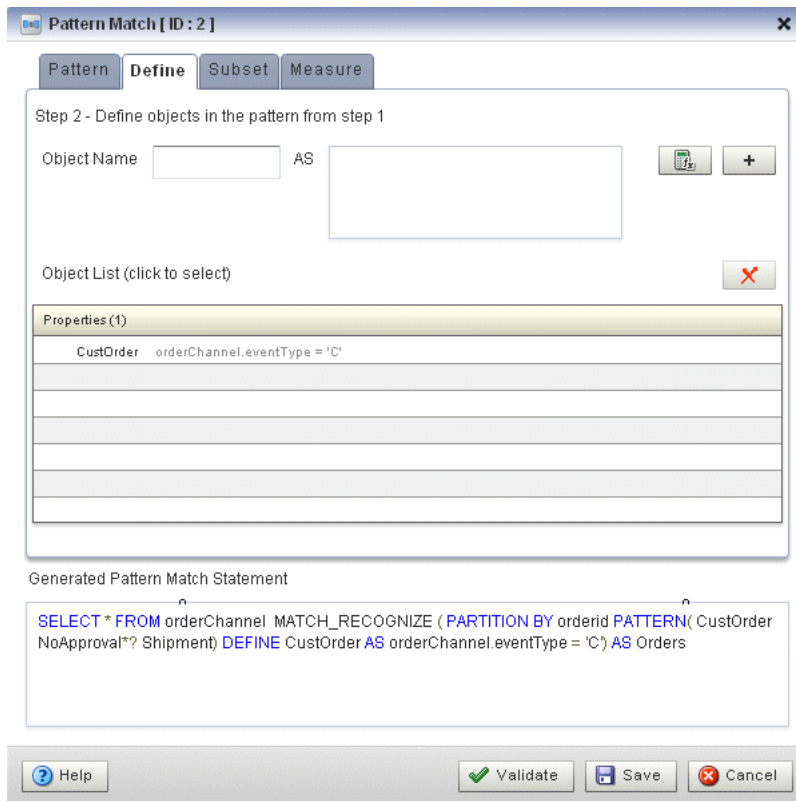
**Figure 2–13 Expression Builder: CustOrder**



**22.** Click **Save**.

**23.** Click the Plus Sign button.

The condition definition is added to the Object List as [Figure 2–14](#) shows.

**Figure 2–14 Pattern Configuration Dialog: Define Tab With CustOrder Condition**

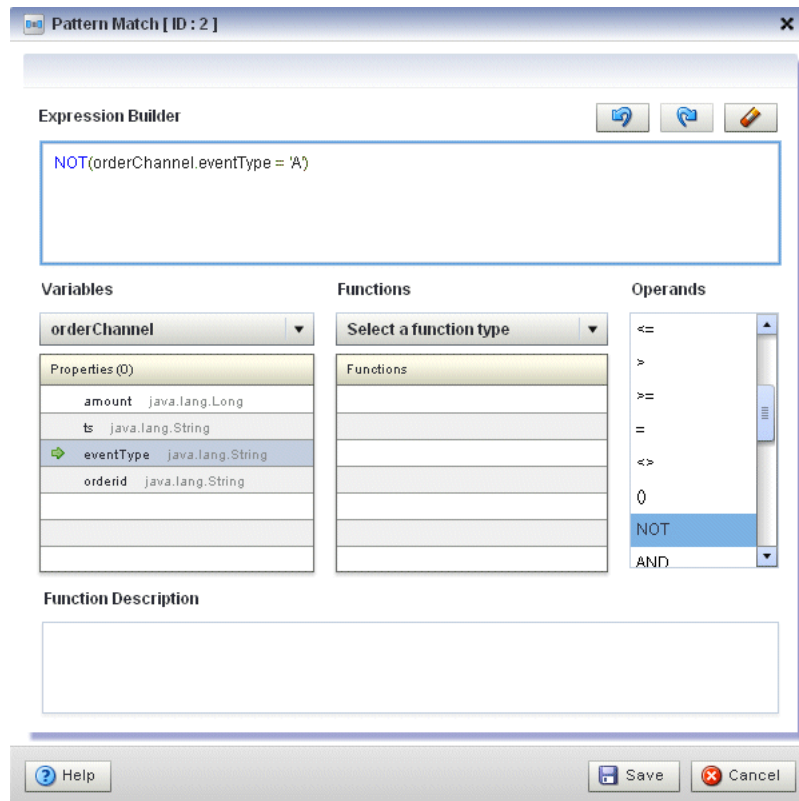
24. Enter **NoApproval** in the **Object Name** field.

25. Click the Expression Builder button and configure the Expression Builder as follows (see [Figure 2–15](#)):

- In the **Variables** list, double-click **eventType**.
- In the **Operands** list, double-click **=**.
- After the **=** operand, enter the value **'A'**.
- Place parenthesis around the expression.
- Place the insertion bar at the beginning of the expression, outside the open parenthesis.
- In the **Operands** list, double-click **NOT**.



Figure 2–15 Expression Builder: NoApproval



26. Click **Save**.

27. Click the Plus Sign button.

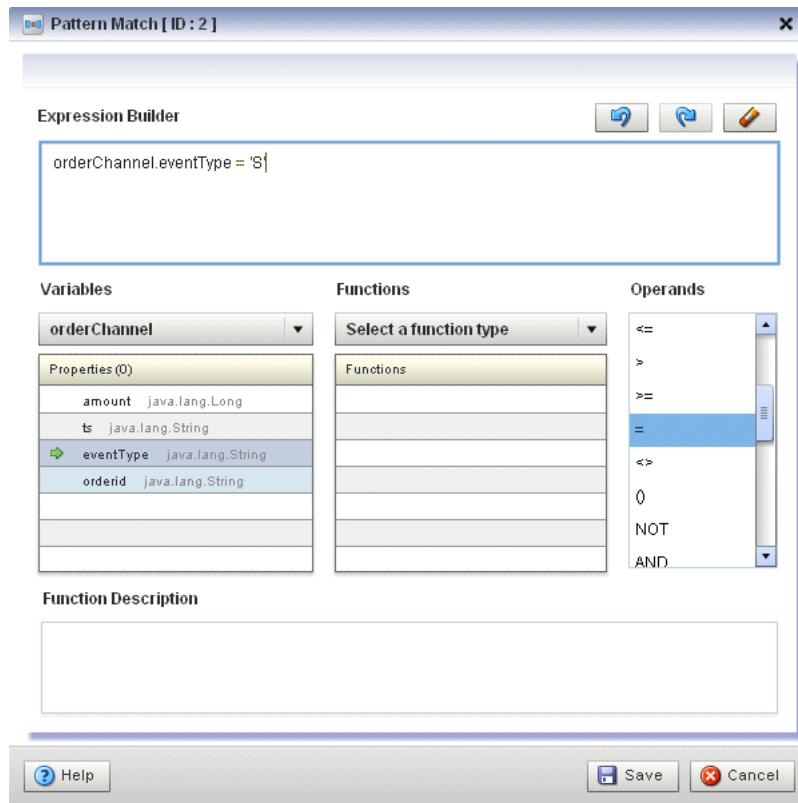
The condition definition is added to the Object List.

28. Enter **Shipment** in the **Object Name** field.

29. Click the Expression Builder button and configure the Expression Builder as follows (see Figure 2–16):

- In the **Variables** list, double-click **eventType**.
- In the **Operands** list, double-click **=**.
- After the = operand, enter the value **'S'**.

**Figure 2–16 Expression Builder: Shipment**



30. Click **Save**.
31. Click the Plus Sign button.  
The Define tab appears as [Figure 2–17](#) shows.

**Figure 2–17 Pattern Configuration Dialog: Define Tab Complete**

Pattern Match [ ID : 2 ]

Pattern Define Subset Measure

Step 2 - Define objects in the pattern from step 1

Object Name  AS

Object List (click to select)

| Properties (3) |                                   |
|----------------|-----------------------------------|
| CustOrder      | orderChannel.eventType = 'C'      |
| NoApproval     | NOT(orderChannel.eventType = 'A') |
| Shipment       | orderChannel.eventType = 'S'      |
|                |                                   |
|                |                                   |
|                |                                   |

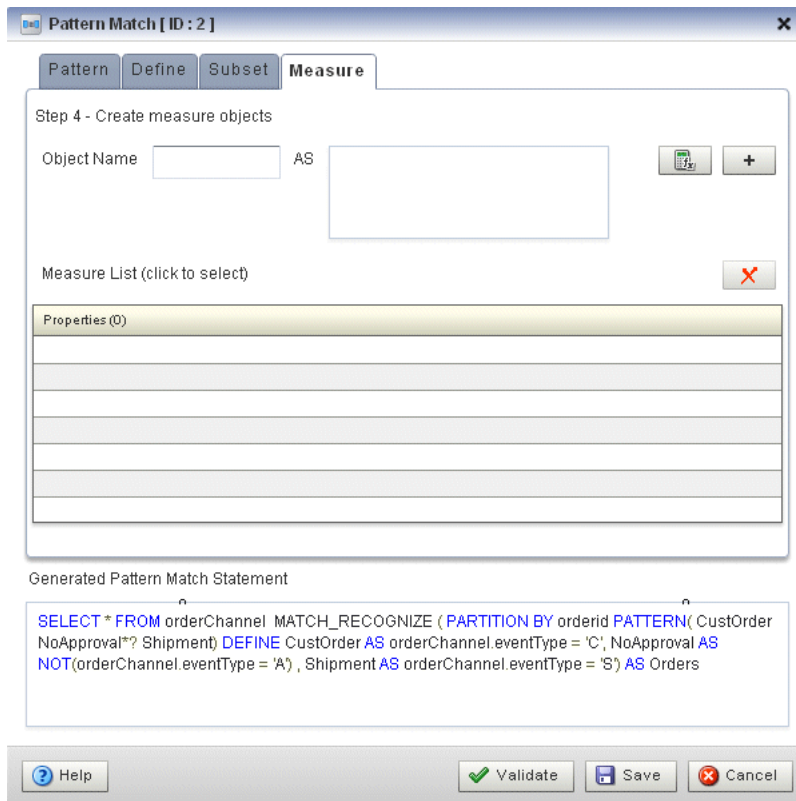
Generated Pattern Match Statement

```
SELECT * FROM orderChannel MATCH_RECOGNIZE ( PARTITION BY orderid PATTERN( CustOrder
NoApproval*? Shipment) DEFINE CustOrder AS orderChannel.eventType = 'C', NoApproval AS
NOT(orderChannel.eventType = 'A') , Shipment AS orderChannel.eventType = 'S') AS Orders
```

Help Validate Save Cancel

**32. Click the Measure tab.**

The Measure tab appears as [Figure 2–18](#) shows.

**Figure 2–18 Measure Tab**

Use the Measure tab to define expressions in a MATCH\_RECOGNIZE condition and to bind stream elements that match conditions in the DEFINE clause to arguments that you can include in the select statement of a query.

Use the Measure tab to specify the following:

- CustOrder.orderid AS orderid
- CustOrder.amount AS amount

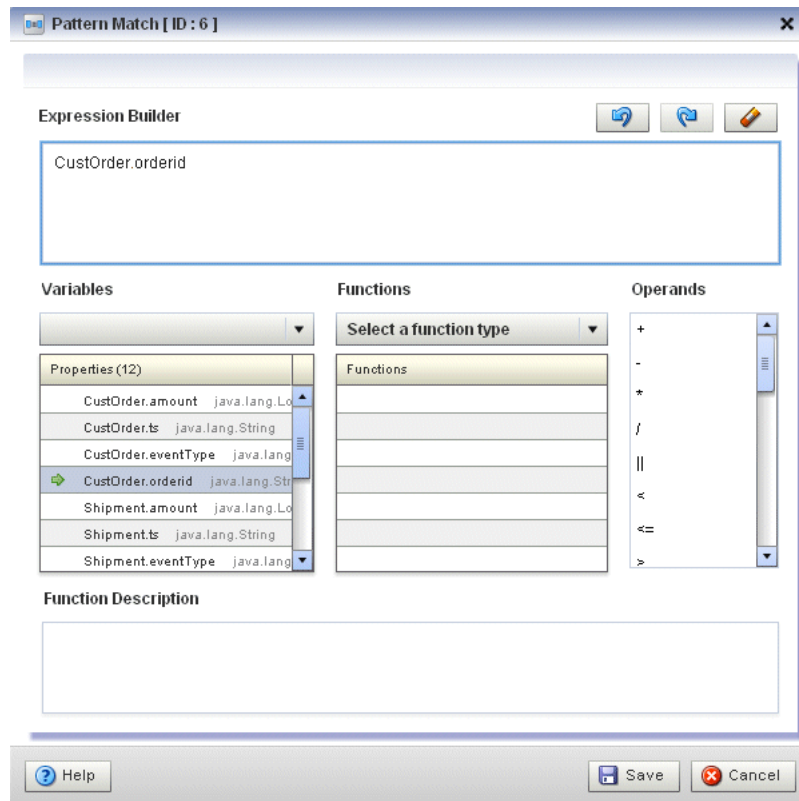
For more information, see:

- "MEASURES Clause" in the *Oracle Fusion Middleware CQL Language Reference for Oracle Event Processing*
- "MATCH\_RECOGNIZE Condition" in the *Oracle Fusion Middleware CQL Language Reference for Oracle Event Processing*

**33.** Enter **orderid** in the **Object Name** field.

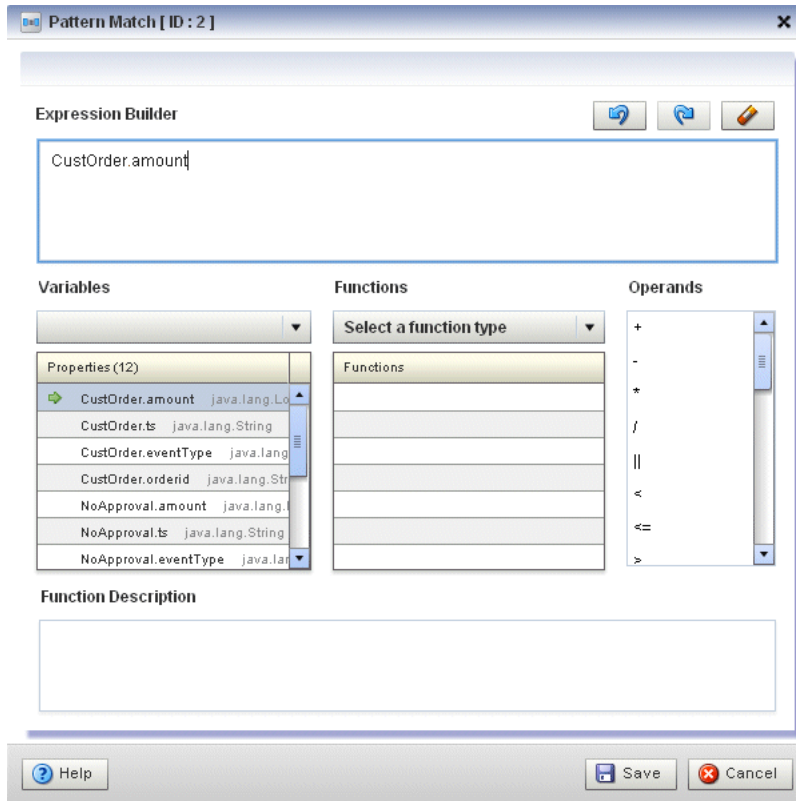
**34.** Click the Expression Builder button and configure the Expression Builder as follows (see [Figure 2–19](#)):

- In the **Variables** list, double-click **CustOrder.orderid**.

**Figure 2–19 Expression Builder: orderid**

35. Click **Save**.
36. Click the Plus Sign button.
37. Enter **amount** in the **Object Name** field.
38. Click the Expression Builder button and configure the Expression Builder as follows (see [Figure 2–20](#)):
  - In the **Variables** list, double-click **CustOrder.amount**.

**Figure 2–20 Expression Builder: amount**



39. Click **Save**.

40. Click the Plus Sign button.

The Measure tab appears as [Figure 2–21](#) shows.

**Figure 2–21 Measure Tab: Complete**

Pattern Match [ ID : 2 ]

Pattern Define Subset **Measure**

Step 4 - Create measure objects

Object Name

Measure List (click to select)

| Properties (2) |                   |
|----------------|-------------------|
| orderid        | CustOrder.orderid |
| amount         | CustOrder.amount  |
|                |                   |
|                |                   |
|                |                   |
|                |                   |

Generated Pattern Match Statement

```
SELECT * FROM orderChannel MATCH_RECOGNIZE ( PARTITION BY orderid MEASURES
CustOrder.orderid AS orderid, CustOrder.amount AS amount PATTERN( CustOrder NoApproval*?
Shipment) DEFINE CustOrder AS orderChannel.eventType = 'C', NoApproval AS
NOT(orderChannel.eventType = 'A'), Shipment AS orderChannel.eventType = 'S') AS Orders
```

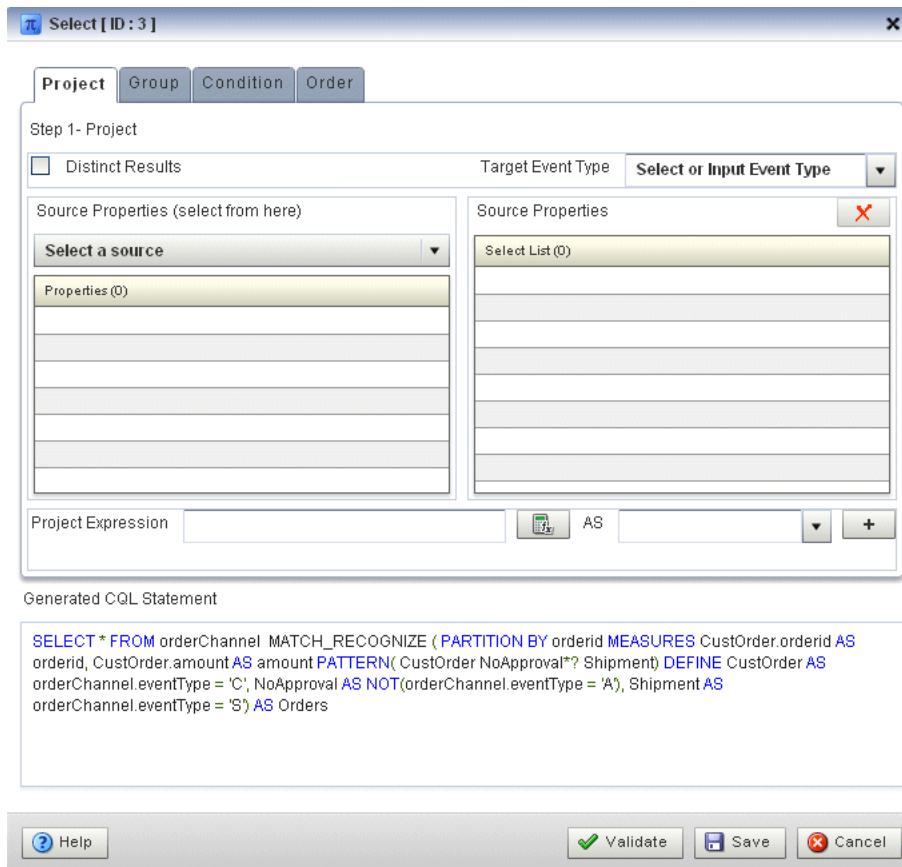
Help Validate Save Cancel

41. Click **Save**.

42. Double-click the **Select** icon.

The Select configuration screen appears as [Figure 2–22](#) shows.

**Figure 2–22 Select Configuration Dialog: Project Tab**



43. Configure the Project tab as follows:
    - Select **AlertEvent** from the **Select or Input Event Type** pull-down menu.
    - Select **Orders** from the **Select a source** pull-down menu.
  44. Double-click **orderid** in the **Properties** list and select **orderid** from the **Select or Input Alias** pull-down menu.
  45. Click the Plus Sign button to add the property to the Generated CQL Statement.
  46. Double-click **amount** in the **Properties** list and select **amount** from the **Select or Input Alias** pull-down menu.
  47. Click the Plus Sign button to add the property to the Generated CQL Statement.
  48. Click in the Project Expression field and enter the value "Error - Missing Approval" and select **alertType** from the **Select or Input Alias** pull-down menu.
  49. Click the Plus Sign button to add the property to the Generated CQL Statement.
- The Project tab appears as [Figure 2–23](#) shows.



**Figure 2–23 Select Configuration Dialog: Project Tab Complete**

Step 1- Project

Distinct Results Target Event Type **AlertEvent**

Source Properties (select from here)

**Orders**

| Properties (2) |                   |
|----------------|-------------------|
| orderid        | CustOrder.orderid |
| amount         | CustOrder.amount  |

Source Properties

| Select List (3)         |
|-------------------------|
| Orders.orderid          |
| Orders.amount           |
| "Error - Missing Event" |

Project Expression  AS

Generated CQL Statement

```
SELECT Orders.orderid AS orderid,Orders.amount AS amount,"Error - Missing Event" AS alertType FROM orderChannel
MATCH_RECOGNIZE ( PARTITION BY orderid MEASURES CustOrder.orderid AS orderid, CustOrder.amount AS
amount PATTERN( CustOrder NoApproval*? Shipment) DEFINE CustOrder AS orderChannel.eventType = 'C',
NoApproval AS NOT(orderChannel.eventType = 'A'), Shipment AS orderChannel.eventType = 'S') AS Orders
```

Help Validate Save Cancel

50. Click **Save**.

51. Click **Save Query**.

52. Double-click the **Output** icon.

The Output configuration screen appears as [Figure 2–24](#) shows.

**Figure 2–24 Output Configuration Dialog**



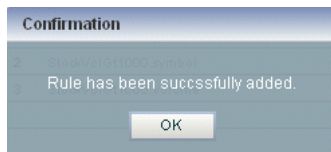
53. Configure the Output as follows:

- Select **Query**.
- Enter **Tracking** as the **Query Name**.

54. Click **Inject Rule**.

The Inject Rule Confirmation dialog appears as [Figure 2–25](#) shows.

**Figure 2–25 Inject Rule Confirmation Dialog**



55. Click **OK**.

The Query Wizard adds the rule to the cq1Proc processor.

56. Click **Save**.

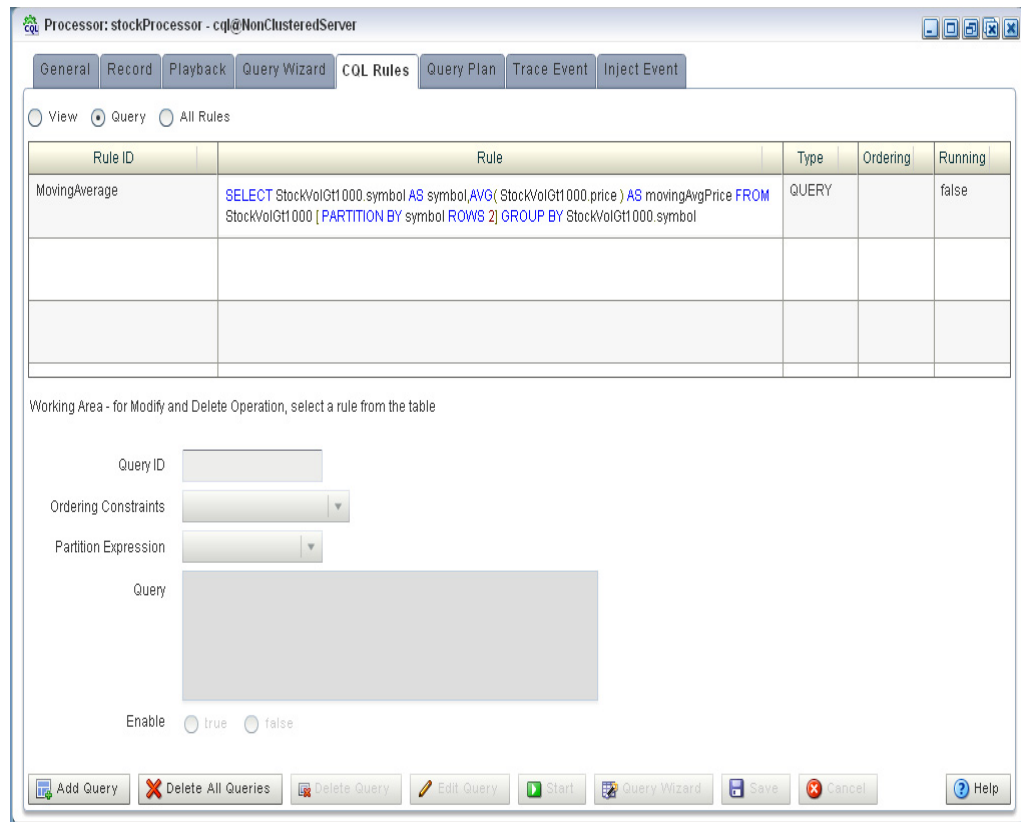
57. Click on the **CQL Rules** tab.

The CQL Rules tab appears as [Figure 2–26](#) shows.

58. Click on the **Query** radio button.

Confirm that your Tracking query is present.

Figure 2–26 CQL Rules Tab With Tracking Query

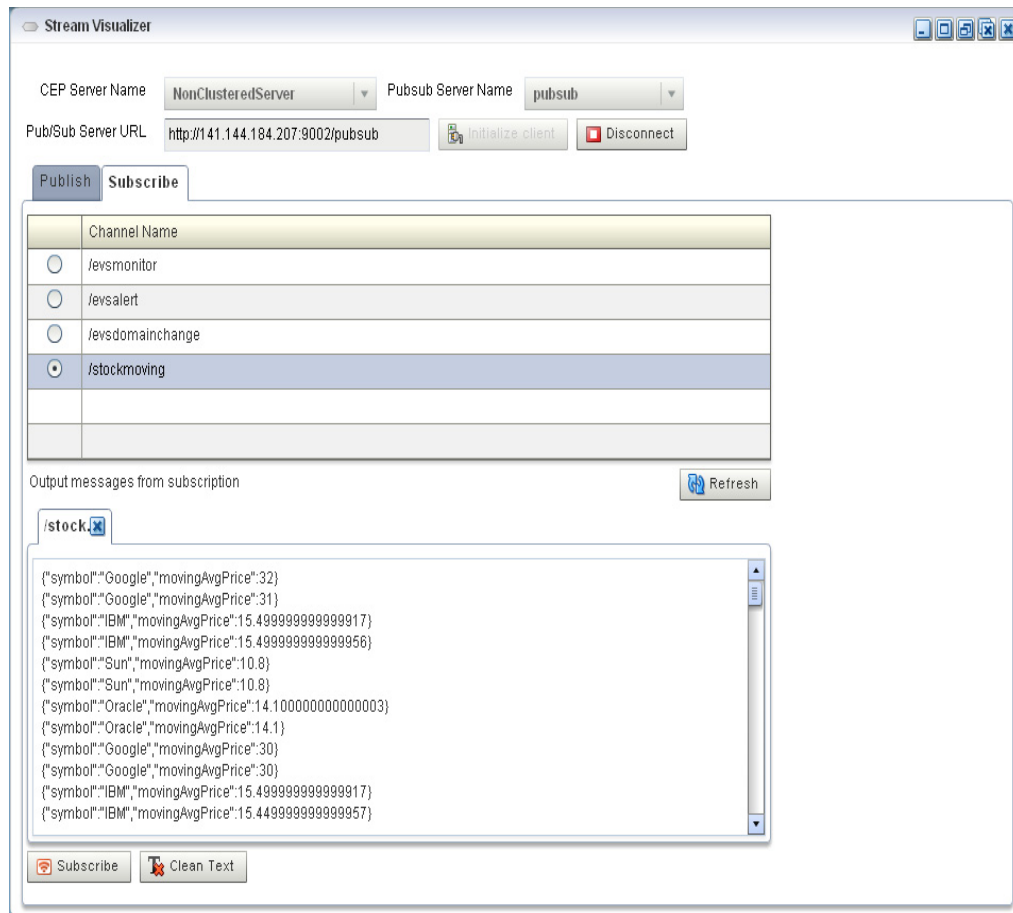
**To test the missing event query:**

1. To simulate the data feed for the missing event query, open a new command window and set your environment as described in [Chapter 3, "Getting Started with Developing Oracle Event Processing Applications"](#)
2. Change to the `MIDDLEWARE_HOME\ocep_11.1\utils\load-generator` directory, where `MIDDLEWARE_HOME` refers to the Middleware directory created when you installed Oracle Event Processing, such as `d:\Oracle\Middleware`.
3. Run the load generator using the `orderData.prop` properties file:
  - a. On Windows:
 

```
prompt> runloadgen.cmd orderData.prop
```
  - b. On UNIX:
 

```
prompt> runloadgen.sh orderData.prop
```
4. In the Oracle Event Processing Visualizer, click the **ViewStream** button in the top pane.

The Stream Visualizer screen appears as [Figure 2–27](#) shows.

**Figure 2–27 Stream Visualizer: Showing Missing Events**

5. Click **Initialize Client**.
6. Click the **Subscribe** tab.
7. Select the **orderalert** radio button.
8. Click **Subscribe**.

As missing events are detected, the Oracle Event Processing updates the **Received Messages** area showing the `AlertEvents` generated.

### Creating the Moving Average Query

This section describes how to use the Oracle Event Processing Visualizer Query Wizard to create the Oracle CQL moving average query that the `stockProc` processor executes.

You do this in two steps:

- First, you create a view (the Oracle CQL equivalent of a subquery) that serves as the source of the moving average query.  
See ["To create a view source for the moving average query:"](#) on page 2-37.
- Second, you create the moving average query using the source view.  
See ["To create the moving average query using the view source:"](#) on page 2-51.
- Finally, you test the moving average query.

See "To test the moving average query:" on page 2-70.

**To create a view source for the moving average query:**

1. If the CQL Oracle Event Processing instance is not already running, follow the procedure in [Section , "Running the CQL Example"](#) to start the server.

You must have a running server to use the Oracle Event Processing Visualizer.

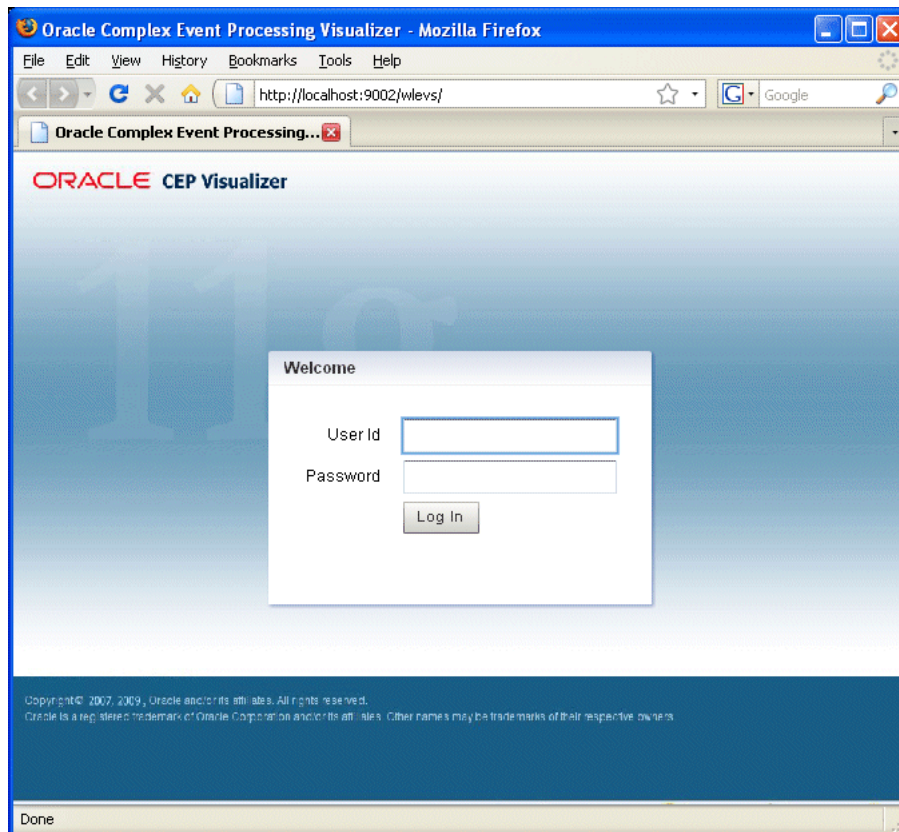
2. Invoke the following URL in your browser:

```
http://host:port/wlevs
```

where *host* refers to the name of the computer on which Oracle Event Processing is running and *port* refers to the Jetty NetIO port configured for the server (default value 9002).

The Logon screen appears as [Figure 2–28](#) shows.

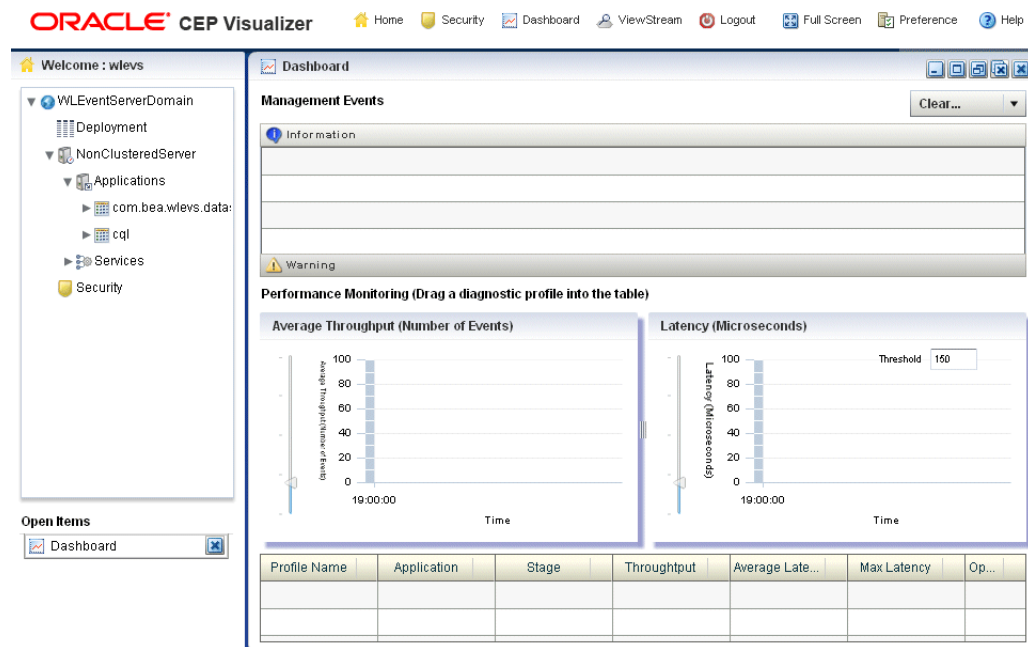
**Figure 2–28 Oracle Event Processing Visualizer Logon Screen**



3. In the Logon screen, enter the **User Id** `wlevs` and **Password** `wlevs`, and click **Log In**.

The Oracle Event Processing Visualizer dashboard appears as [Figure 2–29](#) shows.

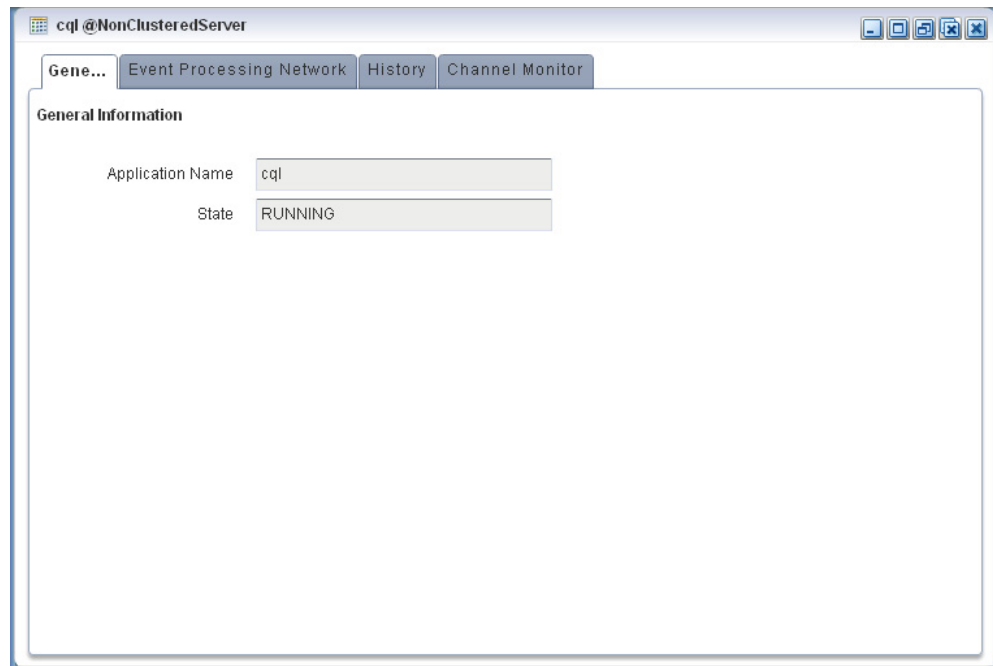
**Figure 2–29 Oracle Event Processing Visualizer Dashboard**



For more information about the Oracle Event Processing Visualizer user interface, see "Understanding the Oracle Event Processing Visualizer User Interface" in the *Oracle Fusion Middleware Visualizer User's Guide for Oracle Event Processing*.

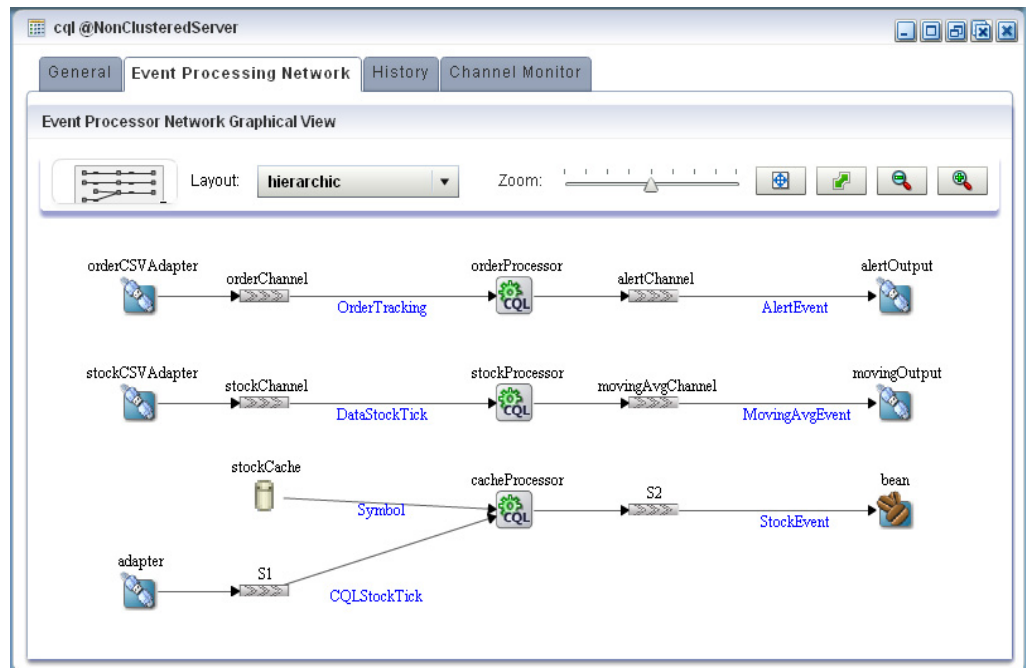
4. In the right-hand pane, expand **WLEventServerDomain** > **NonClusteredServer** > **Applications**.
5. Select the **cql** node.

The CQL application screen appears as [Figure 2–30](#) shows.

**Figure 2–30 CQL Application Screen: General Tab**

6. Select the **Event Processing Network** tab.

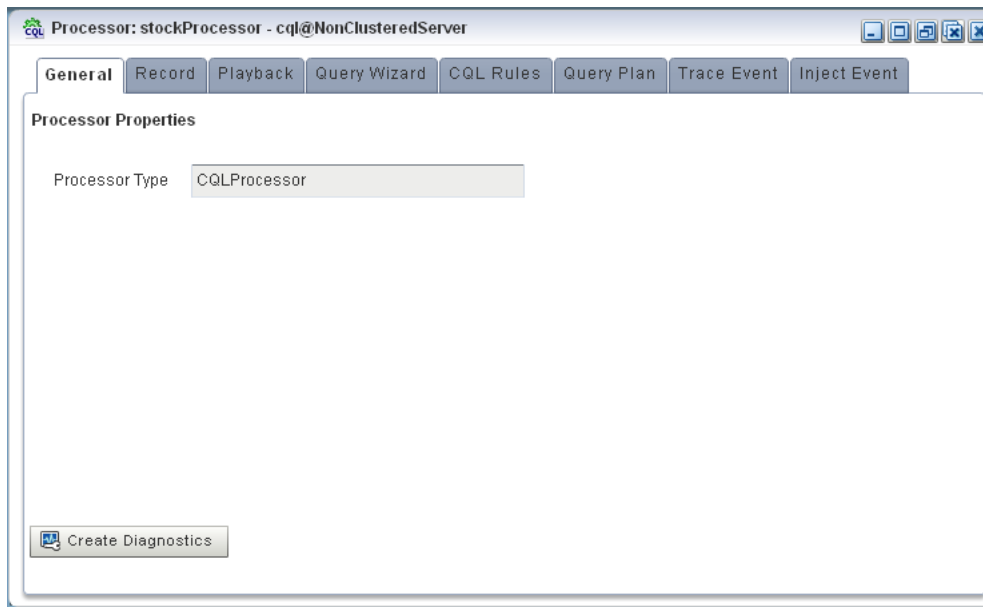
The Event Processing Network screen appears as [Figure 2–31](#) shows.

**Figure 2–31 CQL Application: Event Processing Network Tab**

7. Double-click the **stockProcessor** Oracle CQL processor icon.

The Oracle CQL processor screen appears as [Figure 2–32](#) shows.

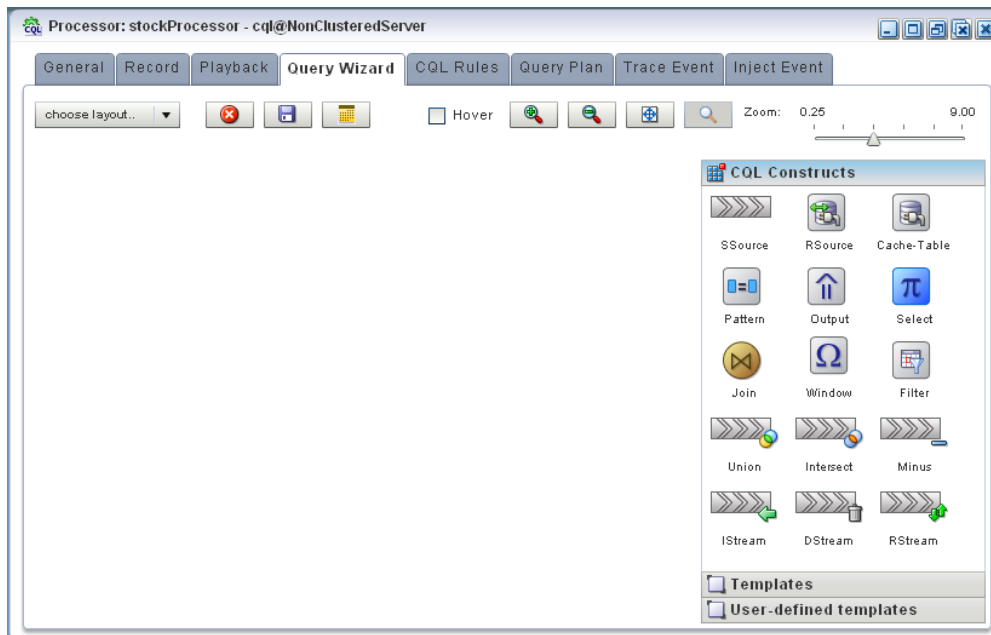
**Figure 2–32 Oracle CQL Processor: General Tab**



8. Select the **Query Wizard** tab.

The Query Wizard screen appears as [Figure 2–33](#) shows.

**Figure 2–33 Oracle CQL Processor: Query Wizard Tab**



You can use the Oracle CQL Query Wizard to construct an Oracle CQL query from a template or from individual Oracle CQL constructs.

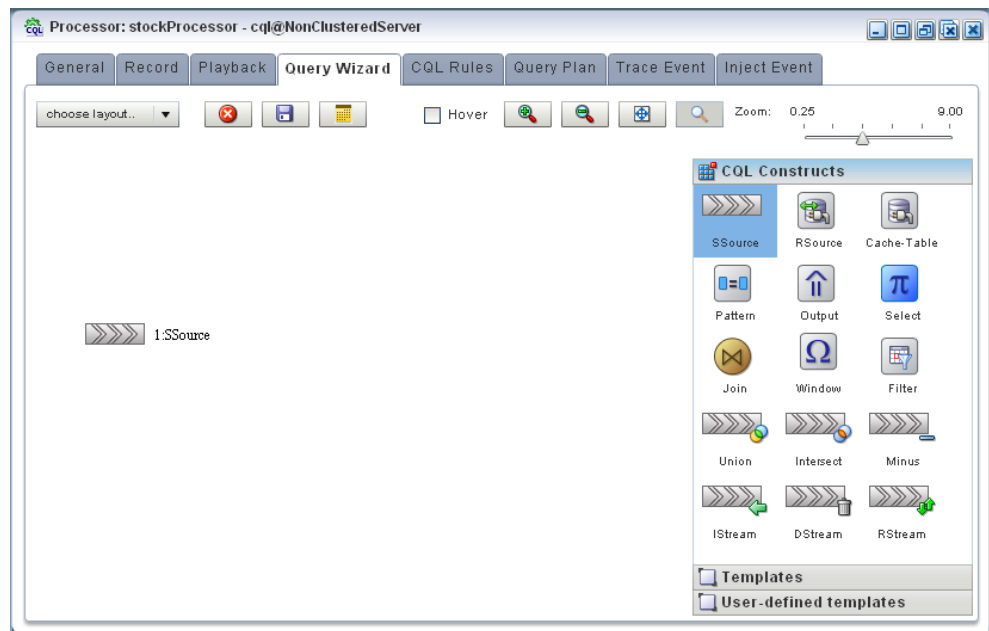


In this procedure, you are going to create an Oracle CQL view and query from individual Oracle CQL constructs.

For more information, see "Creating a Rule in an Oracle CQL Processor Using the Query Wizard" in the *Oracle Fusion Middleware Visualizer User's Guide for Oracle Event Processing*.

9. Click and drag an SSource icon (Stream Source) from the CQL Constructs palette and drop it anywhere in the Query Wizard canvas as [Figure 2-34](#) shows.

**Figure 2-34 Query Wizard: SSource**

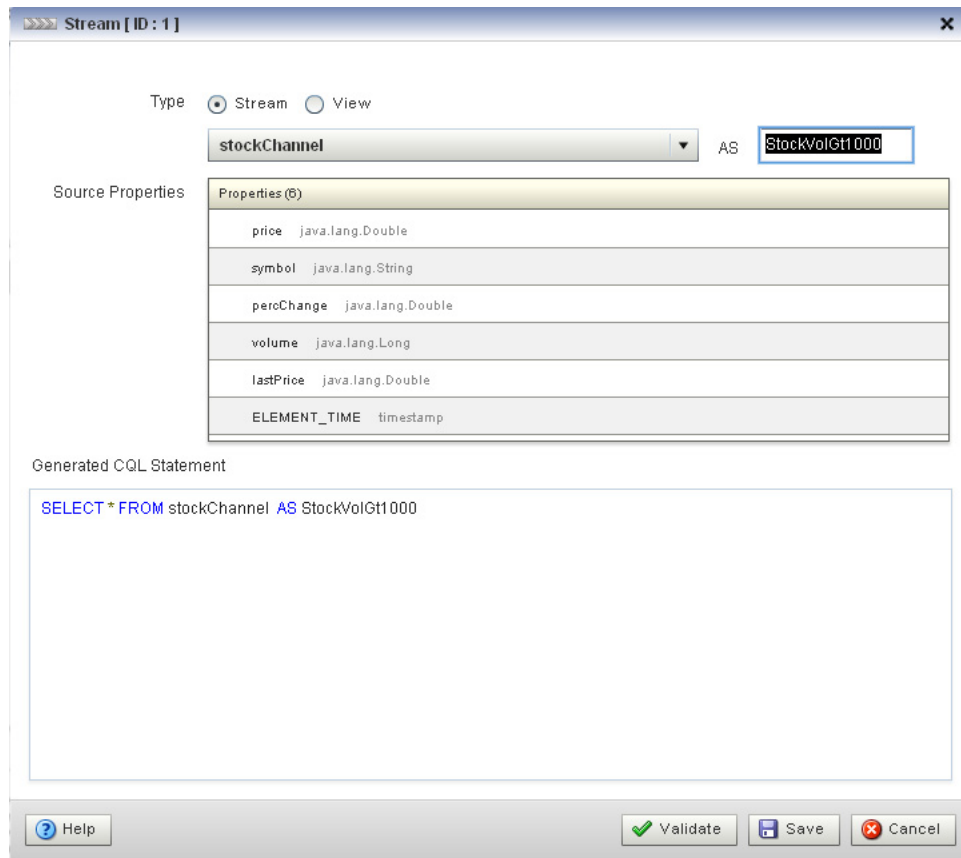


10. Double-click the **SSource** icon.

The SSource configuration screen appears.

The source of your view will be the `stockChannel` stream. You want to select stock events from this stream where the volume is greater than 1000. This will be the source for your moving average query.

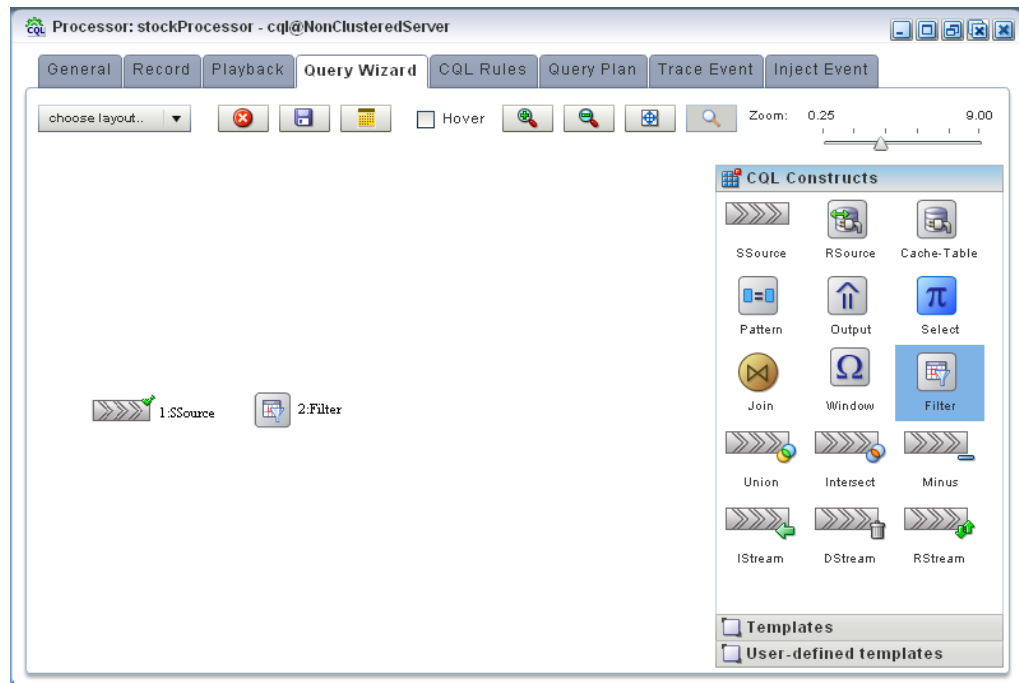
11. Configure the SSource as follows (as shown in [Figure 2-35](#)):
  - Select **Stream** as the Type.  
The source of your view is the `stockChannel` stream.
  - Select **stockChannel** from the **Select a source** pull-down menu.
  - Enter the alias `StockVolGt1000` in the **AS** field.

**Figure 2–35 SSource Configuration Dialog**

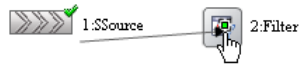
12. Click **Save**.
13. Click **Save Query**.
14. When prompted, enter **StockVolGt1000** in the **Query Id** field.
15. Click **Save**.

Next, you will add an Oracle CQL filter.

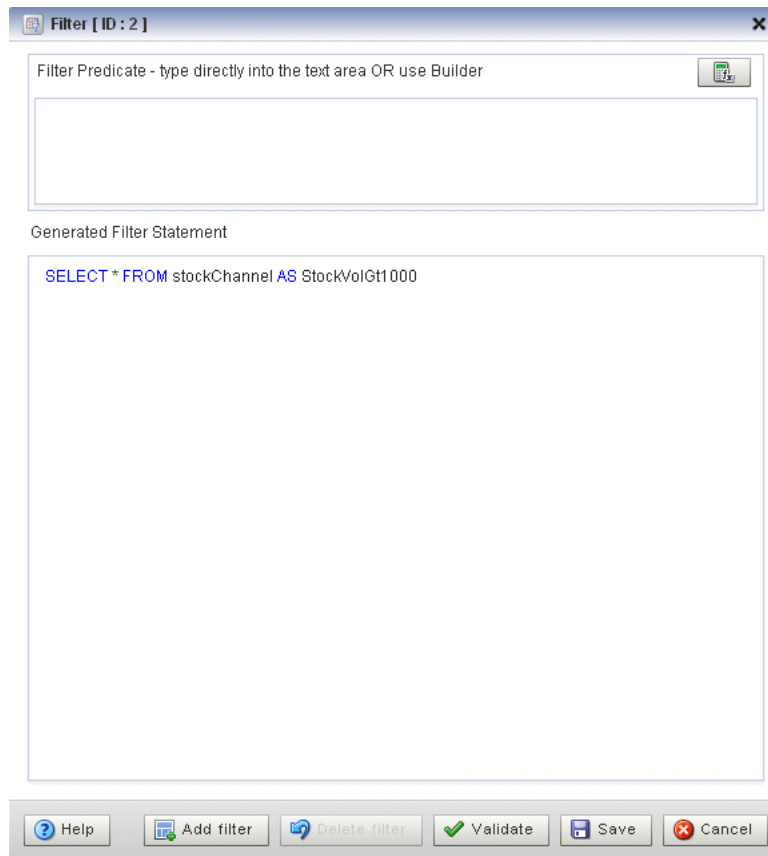
16. Click and drag a Filter icon from the CQL Constructs palette and drop it anywhere in the Query Wizard canvas as [Figure 2–36](#) shows.

**Figure 2–36 Query Wizard: Filter**

17. Click on the SSource icon and drag to the Window icon to connect the Oracle CQL constructs as [Figure 2–37](#) shows.

**Figure 2–37 Connecting the SSource and Filter Icons**

18. Double-click the **Filter** icon.  
The Filter configuration screen appears as [Figure 2–38](#) shows.

**Figure 2–38 Filter Configuration Dialog**

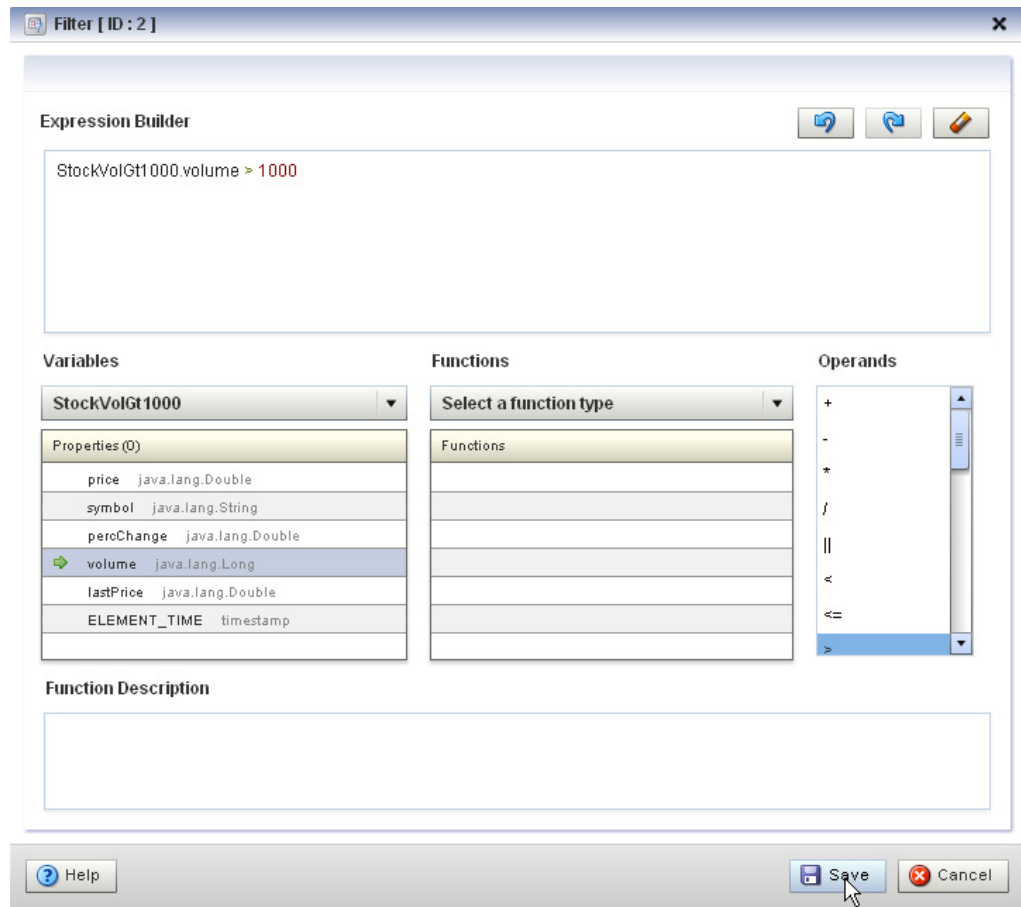
19. Click the Expression Builder button.

The Expression Builder dialog appears.

20. Configure the Expression Builder as follows (as shown in [Figure 2–39](#)):

- Select **StockVolGt100** from the **Select an Event Type** pull-down menu to define the variables you can use in this expression.
- Double-click the **volume** variable to add it to the Expression Builder field.
- Double-click **>** in the **Operands** list to add it to the Expression Builder field.
- Enter the value 1000 after the **>** operand.

Figure 2–39 Filter Expression Builder



21. Click **Save**.

22. Click **Add Filter**.

The Query Wizard adds the expression to the Generated CQL Statement as [Figure 2–40](#) shows.

**Figure 2–40 Filter Configuration Dialog: After Adding the Filter**

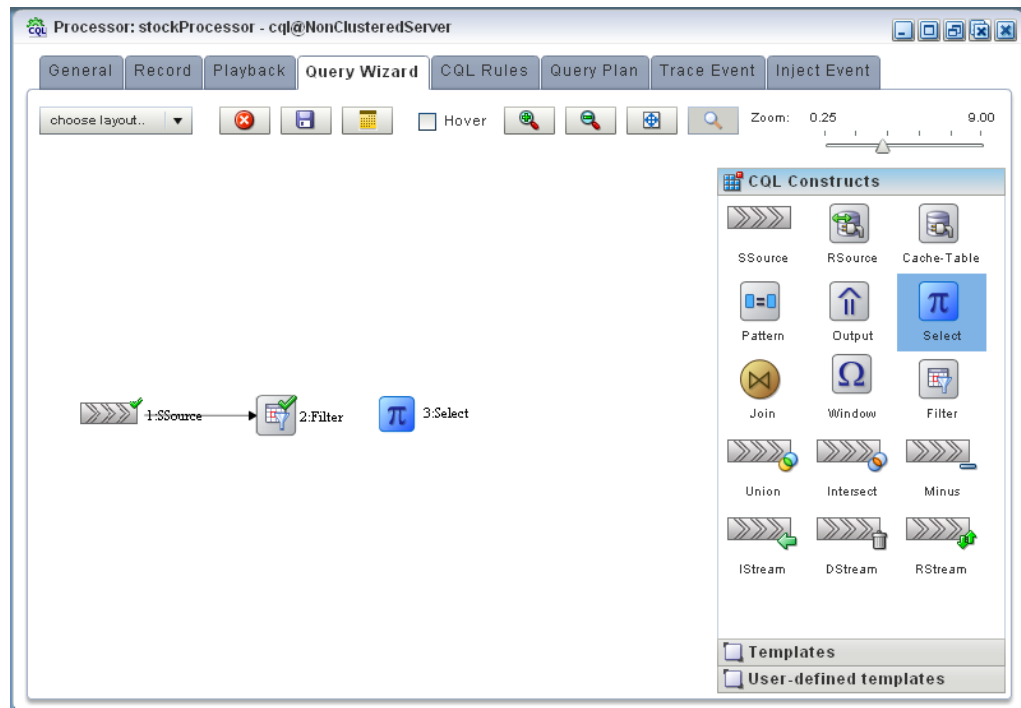
23. Click **Save**.

24. Click **Save Query**.

Next you want to add a select statement.

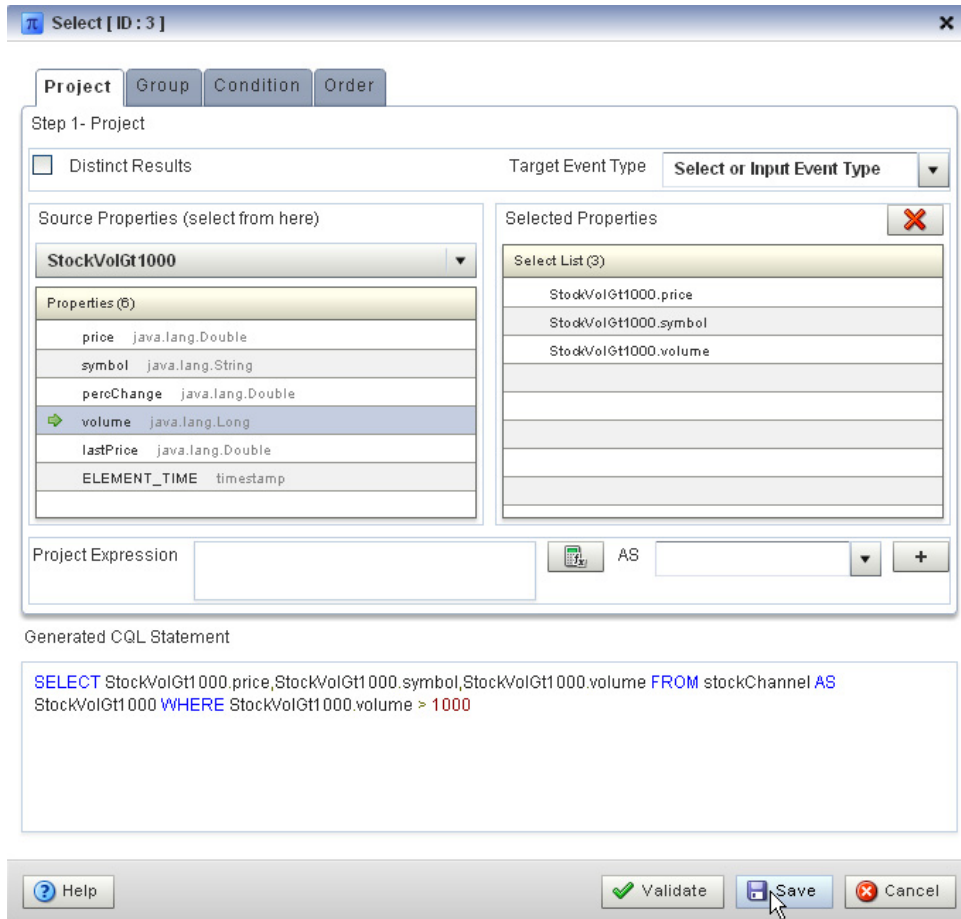
25. Click and drag a Select icon from the CQL Constructs palette and drop it anywhere in the Query Wizard canvas as [Figure 2–41](#) shows.

Figure 2–41 Query Wizard: Select



26. Click on the **Filter** icon and drag to the **Select** icon to connect the Oracle CQL constructs.
27. Double-click the **Select** icon.  
The Select configuration screen appears.  
You want to select price, symbol, and volume from your StockVolGt1000 stream.
28. Configure the Select as follows:
  - Select **StockVolGt1000** from the **Select a source** pull-down menu.
  - Select the **price** property and click the Plus Sign button.  
The Query Wizard adds the property to Generated CQL Statement
  - Repeat for the **symbol** and **volume** properties.
 The Select configuration dialog appears as [Figure 2–42](#) shows.

**Figure 2–42 Select Configuration Dialog: Properties Selected**



29. Click **Save**.

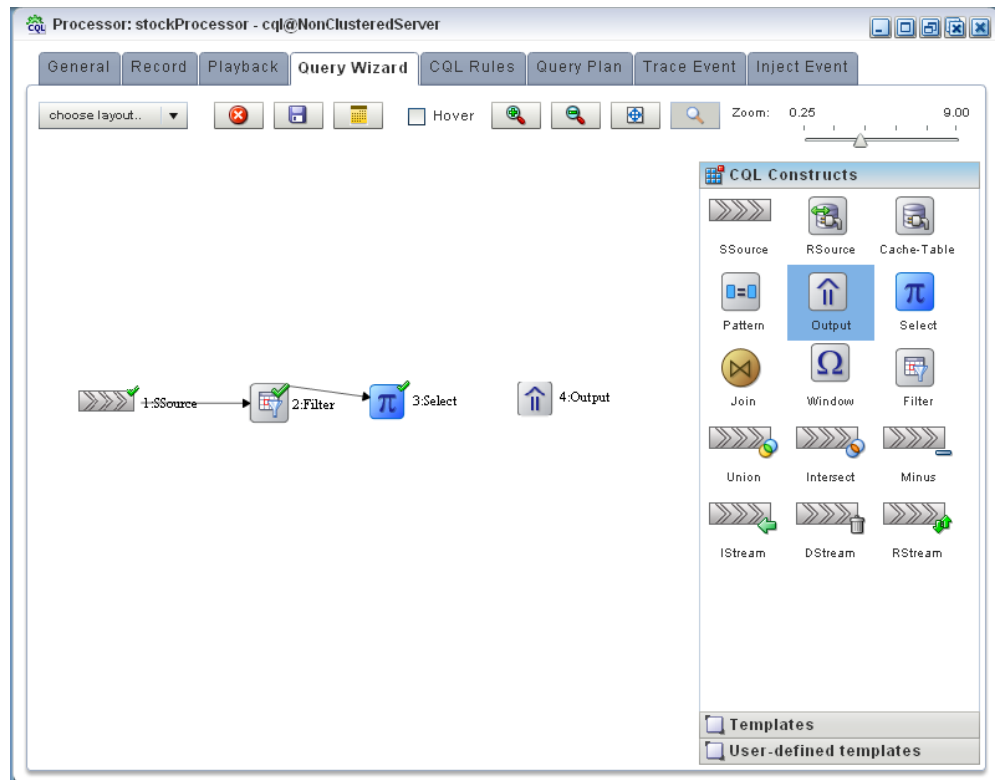
30. Click **Save Query**.

Finally, you will add an Output.

31. Click and drag an Output icon from the CQL Constructs palette and drop it anywhere in the Query Wizard canvas as [Figure 2–43](#) shows.



Figure 2–43 Query Wizard: Output



32. Click on the **Select** icon and drag to the **Output** icon to connect the Oracle CQL constructs.

33. Double-click the **Output** icon.

The Output configuration screen appears.

34. Configure the Output as follows (as shown in [Figure 2–44](#)):

- Select **View**.
- Configure **View Name** as `StockVolGt1000`.
- Delete the contents of the **View Schema** field.

You can let the Oracle Event Processing server define the view schema for you.

**Figure 2–44 Output Configuration Dialog**
**35. Click Inject Rule.**

The Inject Rule Confirmation dialog appears as [Figure 2–45](#) shows.

**Figure 2–45 Inject Rule Confirmation Dialog**
**36. Click OK.**

The Query Wizard adds the rule to the cq1Proc processor.

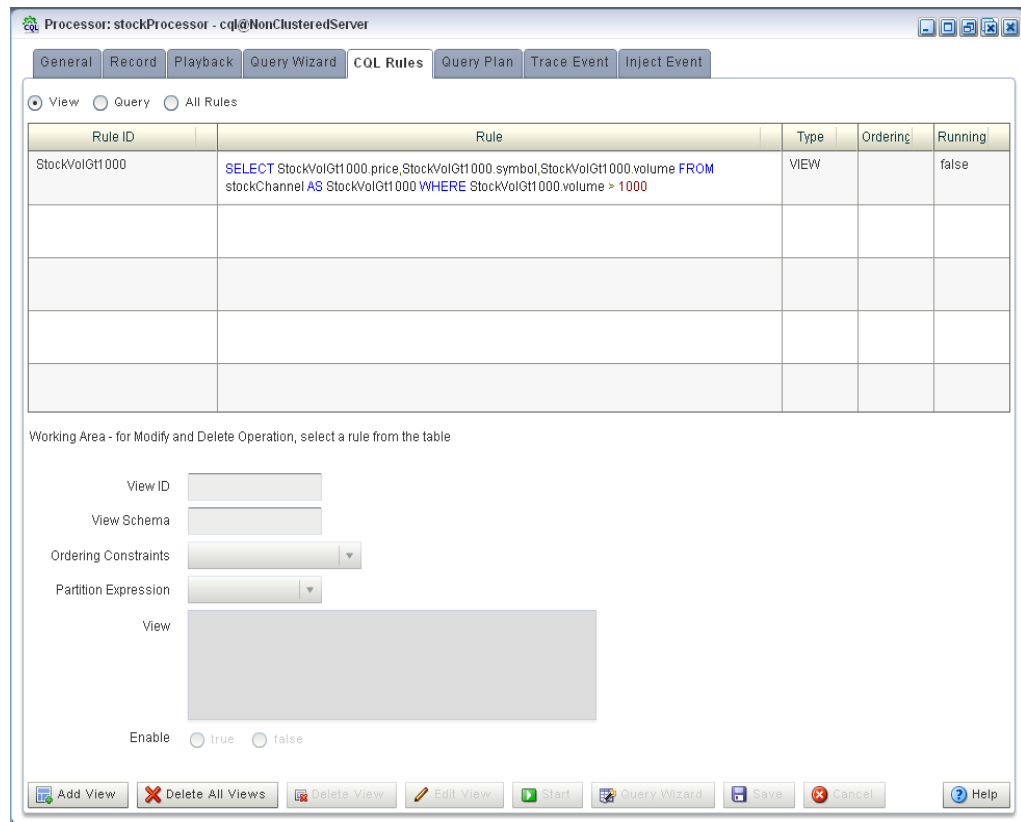
**37. Click Save.****38. Click on the CQL Rules tab.**

The CQL Rules tab appears as [Figure 2–46](#) shows.

**39. Click on the View radio button.**

Confirm that your StockVolGt1000 view is present.

Figure 2–46 CQL Rules Tab With View StockVolGt1000



### To create the moving average query using the view source:

1. If the CQL Oracle Event Processing instance is not already running, follow the procedure in [Section , "Running the CQL Example"](#) to start the server.

You must have a running server to use the Oracle Event Processing Visualizer.

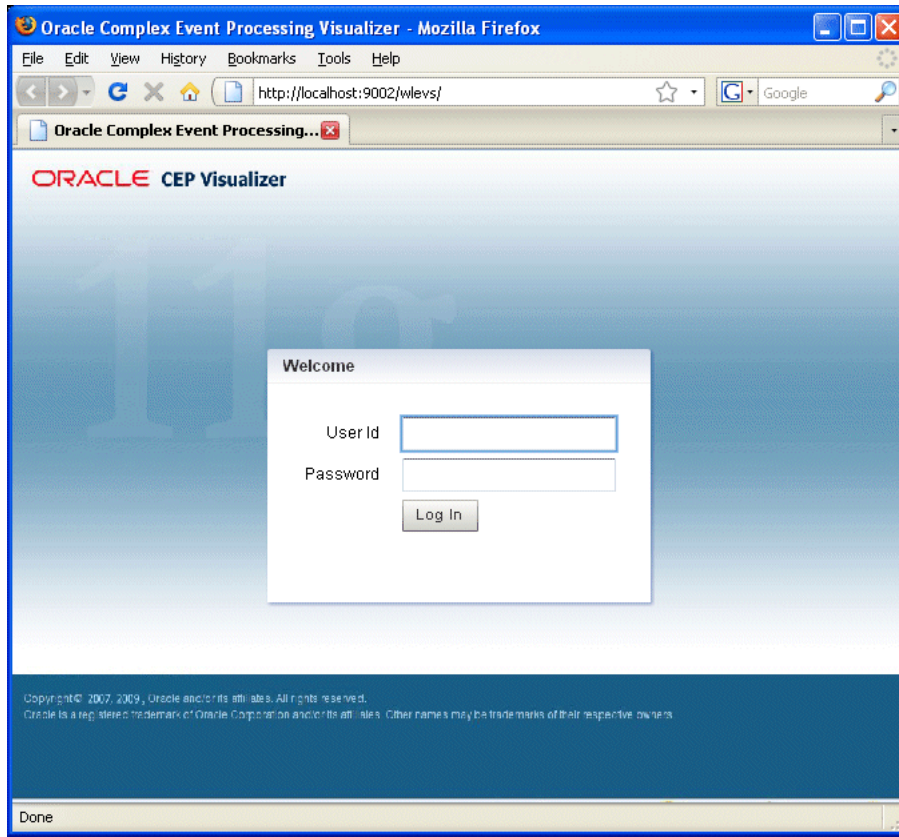
2. Invoke the following URL in your browser:

```
http://host:port/wlevs
```

where *host* refers to the name of the computer on which Oracle Event Processing is running and *port* refers to the Jetty NetIO port configured for the server (default value 9002).

The Logon screen appears as [Figure 2–47](#) shows.

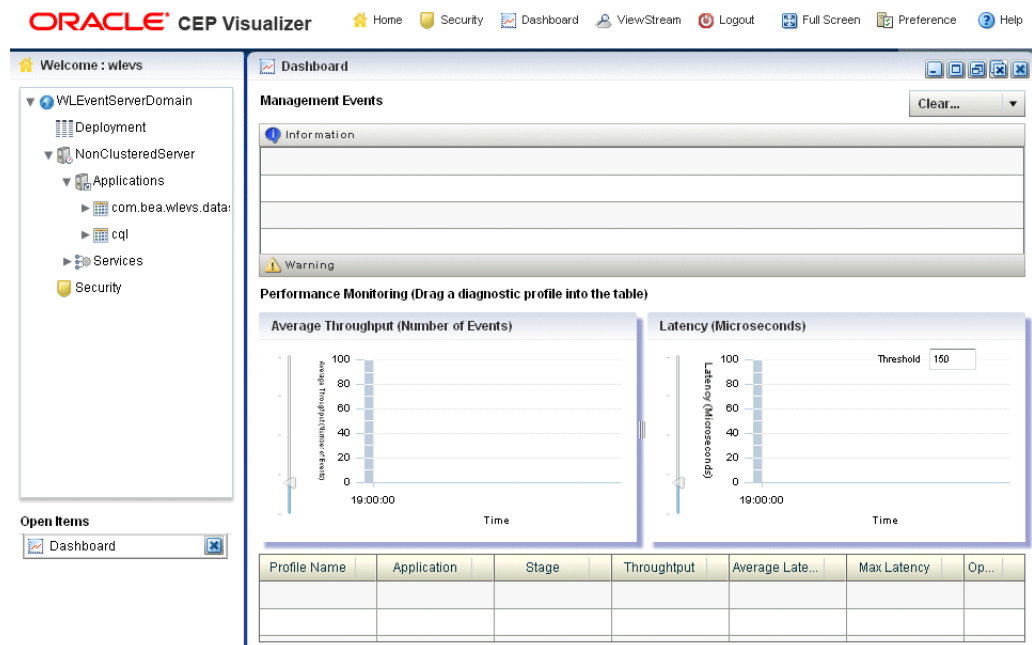
**Figure 2–47 Oracle Event Processing Visualizer Logon Screen**



3. In the Logon screen, enter the **User Id** wlevs and **Password** wlevs, and click **Log In**.

The Oracle Event Processing Visualizer dashboard appears as [Figure 2–48](#) shows.

Figure 2–48 Oracle Event Processing Visualizer Dashboard

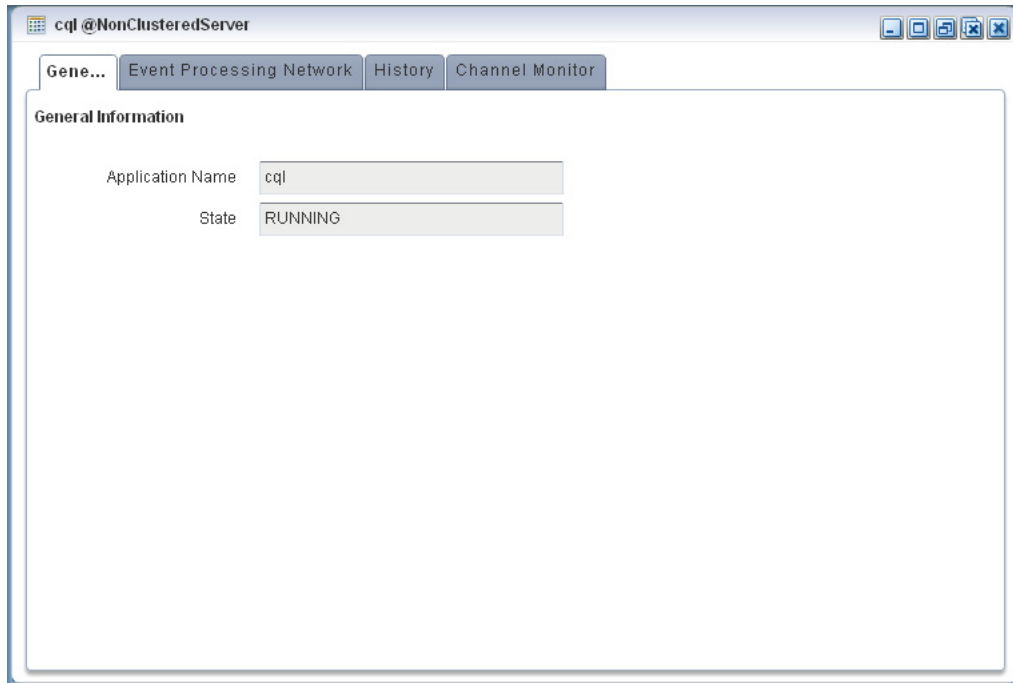


For more information about the Oracle Event Processing Visualizer user interface, see "Understanding the Oracle Event Processing Visualizer User Interface" in the *Oracle Fusion Middleware Visualizer User's Guide for Oracle Event Processing*.

4. In the left-hand pane, expand **WLEventServerDomain** > **NonClusteredServer** > **Applications**.
5. Select the **cql** node.

The CQL application screen appears as [Figure 2–49](#) shows.

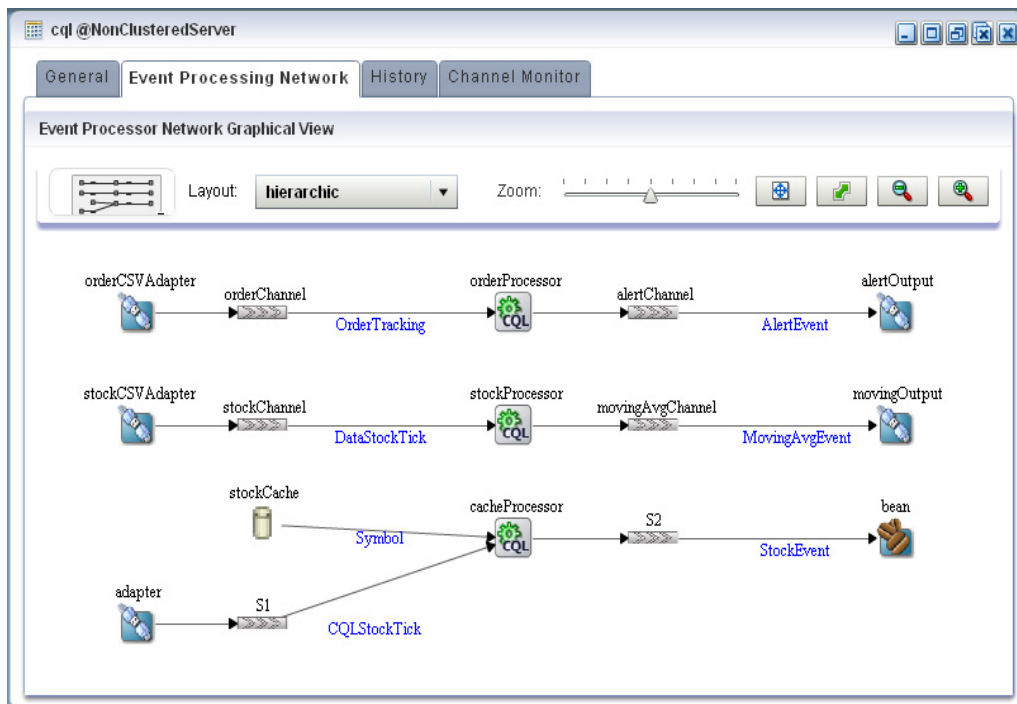
**Figure 2–49 CQL Application Screen: General Tab**



6. Select the **Event Processing Network** tab.

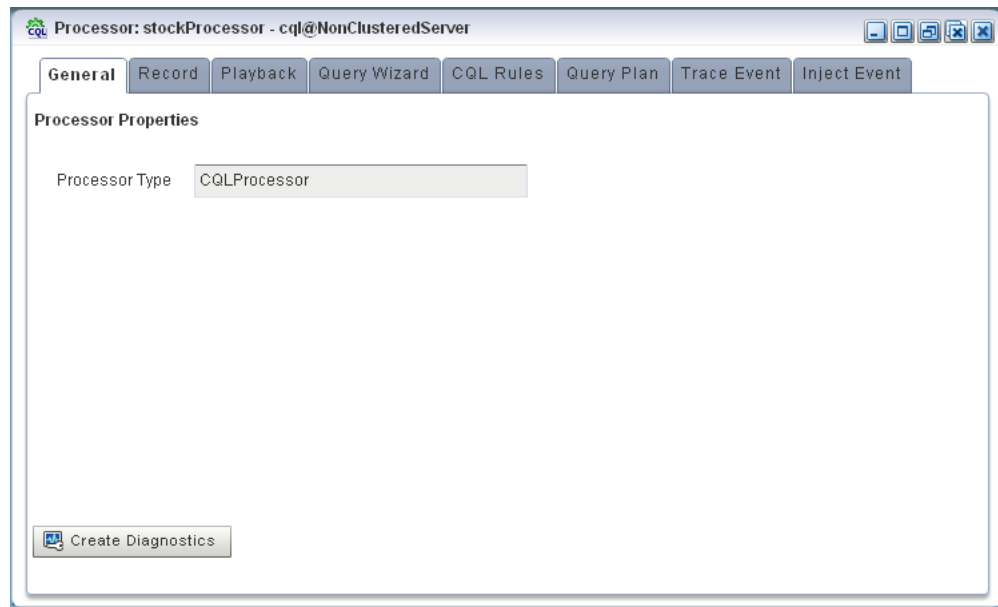
The Event Processing Network screen appears as [Figure 2–50](#) shows.

**Figure 2–50 CQL Application: Event Processing Network Tab**



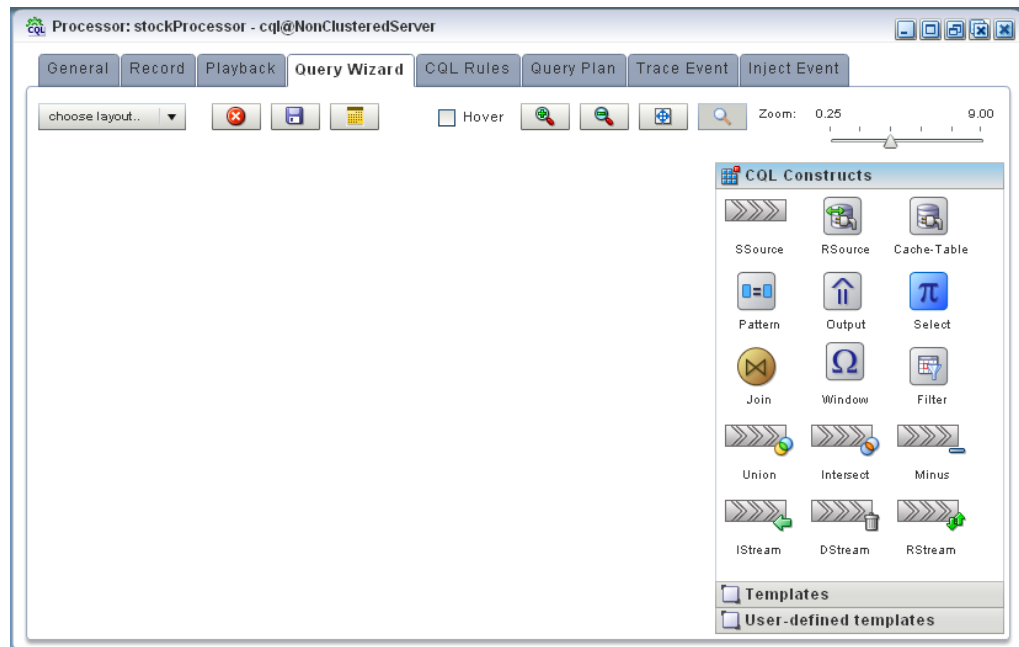
7. Double-click the **stockProcessor** Oracle CQL processor icon.

The Oracle CQL processor screen appears as [Figure 2–51](#) shows.

**Figure 2–51 Oracle CQL Processor: General Tab**

8. Select the **Query Wizard** tab.

The Query Wizard screen appears as [Figure 2–52](#) shows. If you have been recently creating or editing queries for this processor, you might see those queries on the Query Wizard canvas. Otherwise, the canvas will be blank.

**Figure 2–52 Oracle CQL Processor: Query Wizard Tab**

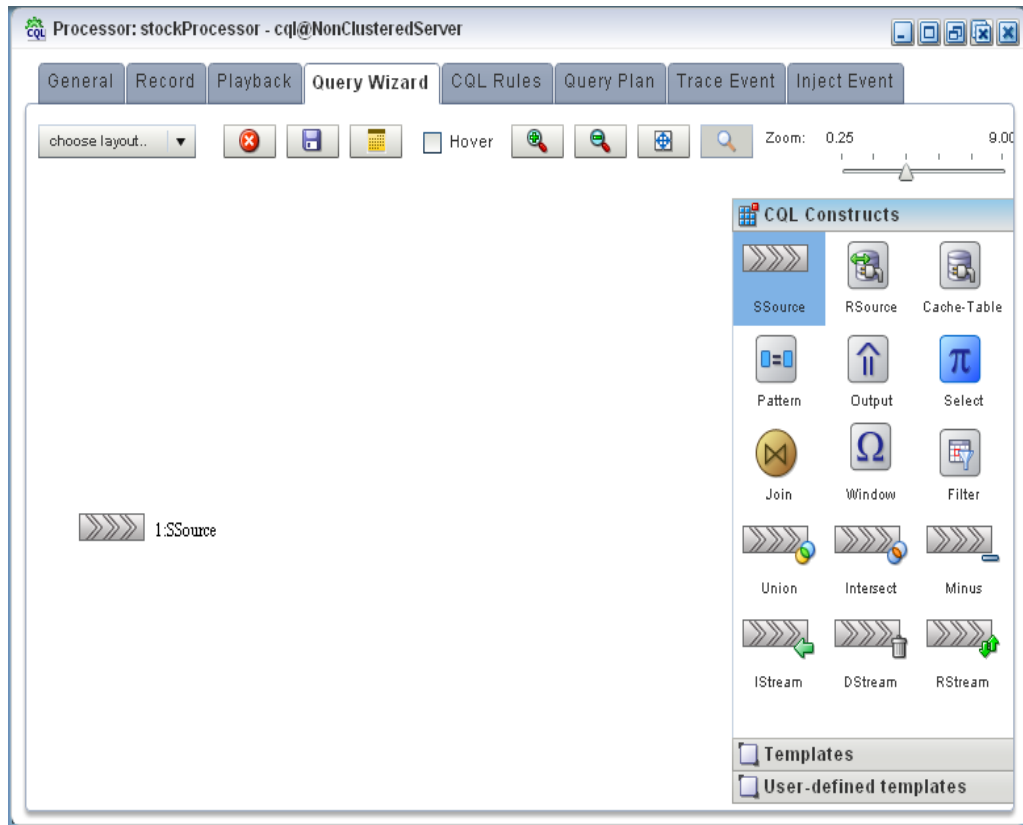
You can use the Oracle CQL Query Wizard to construct an Oracle CQL query from a template or from individual Oracle CQL constructs.

In this procedure, you are going to create an Oracle CQL view and query from individual Oracle CQL constructs.

For more information, see "Creating a Rule in an Oracle CQL Processor Using the Query Wizard" in the *Oracle Fusion Middleware Visualizer User's Guide for Oracle Event Processing*.

9. Click and drag an SSource icon (Stream Source) from the CQL Constructs palette and drop it anywhere in the Query Wizard canvas as [Figure 2–53](#) shows.

**Figure 2–53 Query Wizard: SSource for Moving Average Query**



10. Double-click the **SSource** icon.  
The SSource configuration screen appears.
11. Configure the SSource dialog as follows (as shown in [Figure 2–54](#)):
  - Select **View** as the **Type**.
  - Select the **StockVolGt1000** view from the **Select a source** pull-down menu.



**Figure 2–54 SSource Configuration Dialog: Moving Average Query**

Stream [ID: 1]

Type  Stream  View

StockVolGt1000 AS

Source Properties

| Properties (4) |                  |
|----------------|------------------|
| price          | double           |
| symbol         | java.lang.String |
| volume         | long             |
| ELEMENT_TIME   | timestamp        |
|                |                  |
|                |                  |

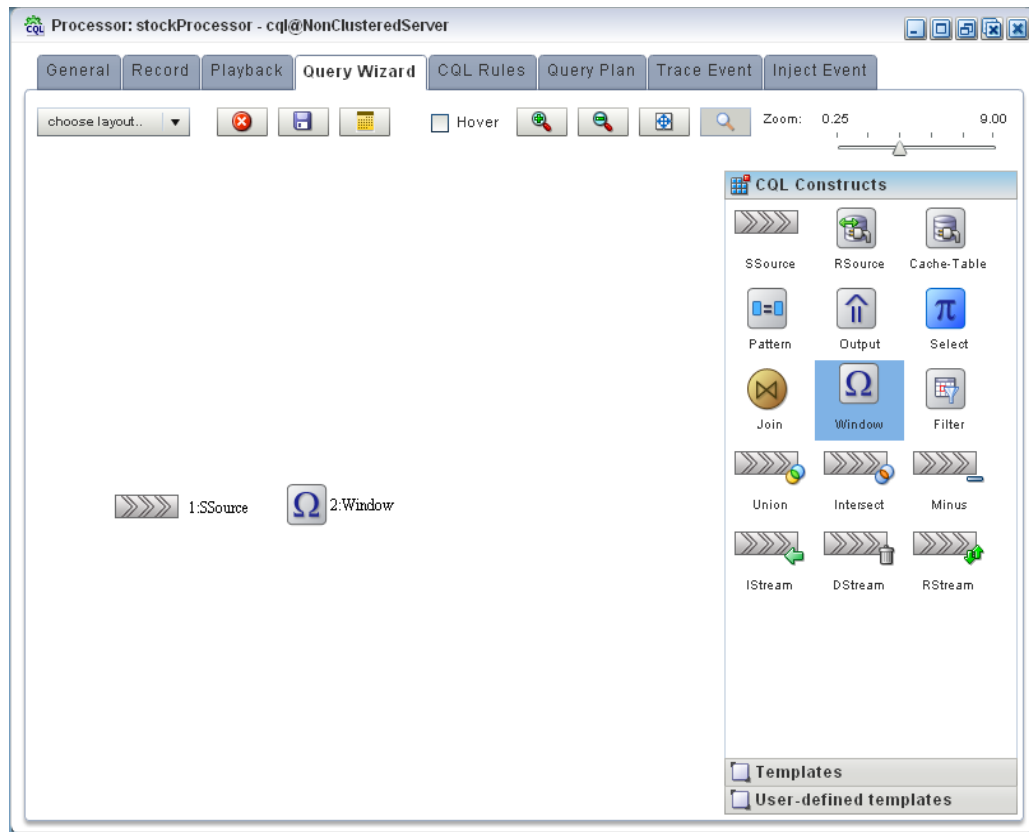
Generated CQL Statement

```
SELECT * FROM StockVolGt1000
```

Help Validate Save Cancel

12. Click **Save**.
13. Click **Save Query**.
14. Click and drag a Window icon from the CQL Constructs palette and drop it anywhere in the Query Wizard canvas as [Figure 2–55](#) shows.

Figure 2–55 Query Wizard: Window for Moving Average Query



15. Click on the **SSource** icon and drag to the **Window** icon to connect the Oracle CQL constructs.
16. Double-click the **Window** icon.  
The SSource configuration screen appears.  
You want to create a sliding window over the last 2 events, partitioned by `symbol`.
17. Configure the Window dialog as follows (as shown in [Figure 2–56](#)):
  - Select **symbol** in the **Source Property List** to add it to the **Partition List**.
  - Select **Partition** as the **Type**.
  - Select **Row Based** and enter 2 in the **Row Based** field.
18. Click Add Window.  
The Query Wizard adds the sliding window to the Generated CQL Statement as [Figure 2–56](#) shows.

**Figure 2–56 Window Configuration Dialog: After Adding Window**

Window [ ID : 2 ]

Partition Source Property List Partition List (select from the list)

price  
symbol

symbol

Type  Now  Time  Row  Partition  Unbounded

Row Based 2

Time Based

ns  micros  ms  sec  min  hour  day

Slide  Row Based

Time Based

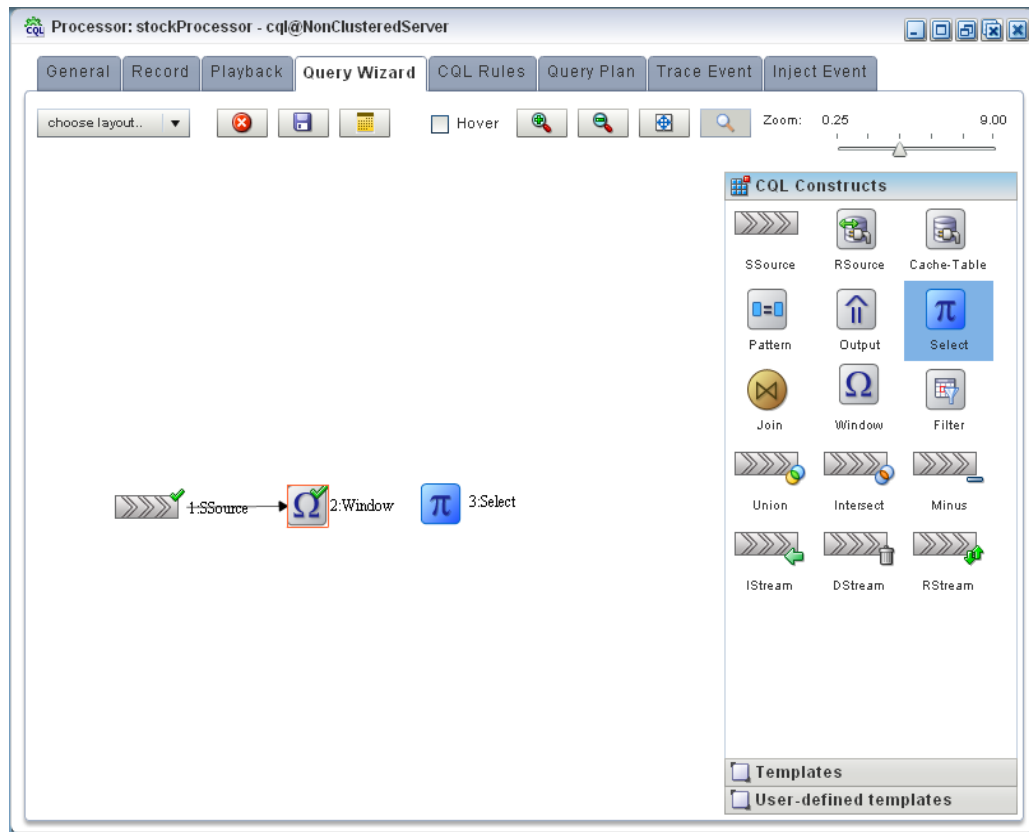
ns  micros  ms  sec  min  hour  day

Generated CQL Statement

```
SELECT * FROM StockVolGt1000 [PARTITION BY symbol ROWS 2]
```

Help Add Window Validate Save Cancel

19. Click **Save**.
20. Click **Save Query**.
21. Click and drag a Select icon from the CQL Constructs palette and drop it anywhere in the Query Wizard canvas as [Figure 2–57](#) shows.

**Figure 2–57 Query Wizard: Select for Moving Average Query**

22. Click on the **Window** icon and drag to the **Select** icon to connect the Oracle CQL constructs.
23. Double-click the **Select** icon.  
The Select configuration screen appears.
24. Select **StockVolGt1000** from the **Select a source** pull-down menu.  
This is the source of moving average query: the view you created earlier (see "[To create a view source for the moving average query:](#)" on page 2-37).
25. Select **MovingAvgEvent** from the **Target Event Type** pull-down menu.  
This is the output event your moving average query will produced. You will map properties from the source events to this output event.
26. In the Source Properties list, select **symbol**.  
The selected source property is added to the Project Expression as [Figure 2–58](#) shows.

**Figure 2–58 Select Configuration Dialog: Source Property symbol Selected**

The screenshot shows a dialog window titled "Select [ ID : 3 ]" with tabs for "Project", "Group", "Condition", and "Order". The "Project" tab is active, showing "Step 1- Project".

Options include "Distinct Results" (unchecked) and "Target Event Type" set to "MovingAvgEvent".

Under "Source Properties (select from here)", a dropdown menu shows "StockVolGt1000". Below it, a list of properties is shown:

| Properties (4) |                  |
|----------------|------------------|
| price          | double           |
| symbol         | java.lang.String |
| volume         | long             |
| ELEMENT_TIME   | timestamp        |

The "symbol" property is highlighted with a green arrow. The "Selected Properties" list on the right is empty, labeled "Select List (0)".

The "Project Expression" field contains "StockVolGt1000.symbol". The "AS" field is empty.

The "Generated CQL Statement" field contains the following SQL:

```
SELECT * FROM StockVolGt1000 [PARTITION BY symbol ROWS 2]
```

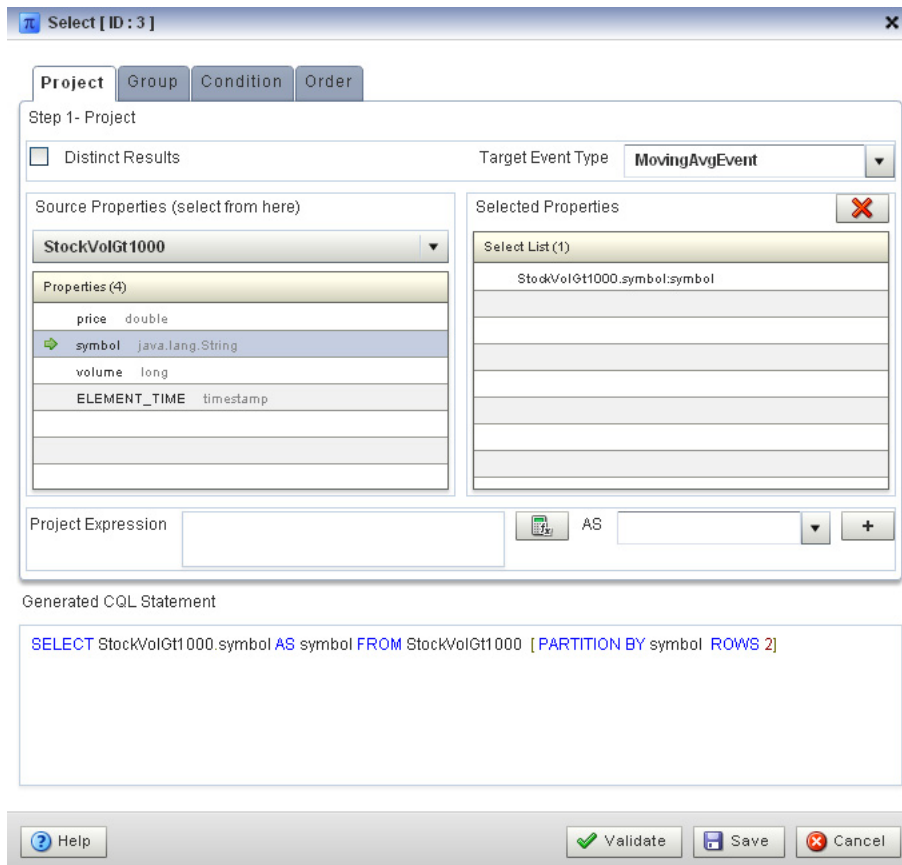
At the bottom, there are buttons for "Help", "Validate", "Save", and "Cancel".

In this case, you just want to map the source property `symbol` to output event property `symbol` as is.

27. Click on the pull-down menu next to the **AS** field and select **symbol**.
28. Click the Plus Sign button.

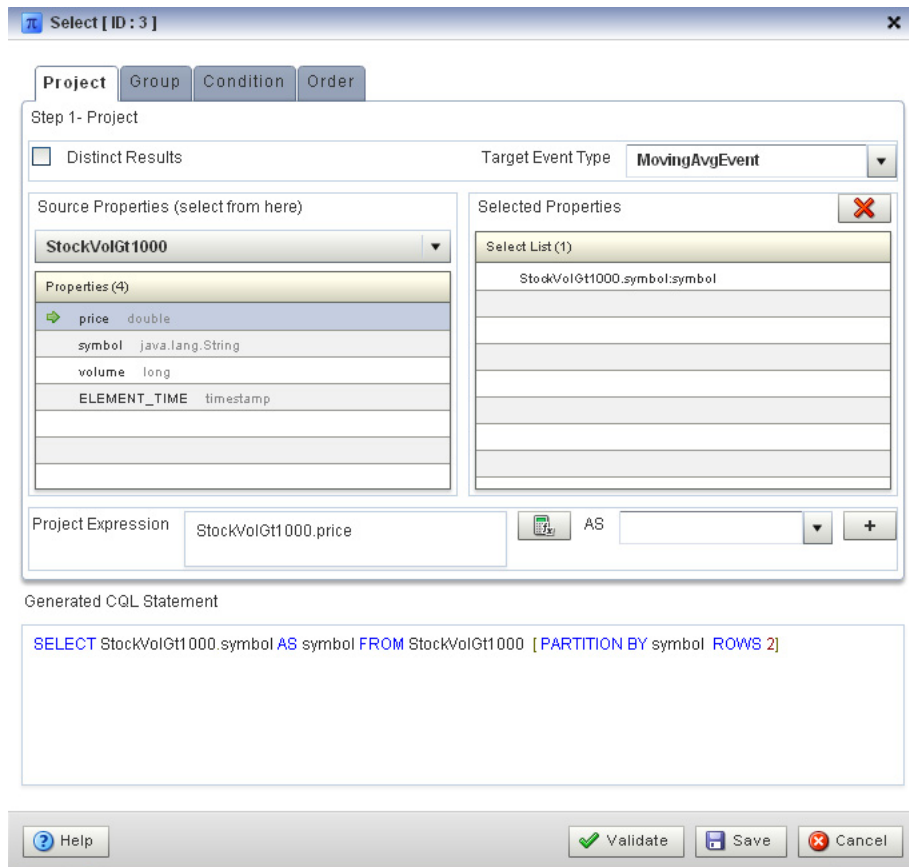
The source property is added to the project expression of the Generated CQL Statement as [Figure 2–59](#) shows.

**Figure 2–59 Select Configuration Dialog: Source Property symbol Mapped to Output Event Property**



29. In the Source Properties list, select **price**.

The selected source property is added to the Project Expression as [Figure 2–60](#) shows.

**Figure 2–60 Select Configuration Dialog: Source Property price Selected**

In this case, you want to process the source property `price` before you map it to the output event.

30. Click the Expression Builder button.

The Expression Builder dialog appears.

31. Select **Aggregate Function** from the **Select a function type** pull-down menu.

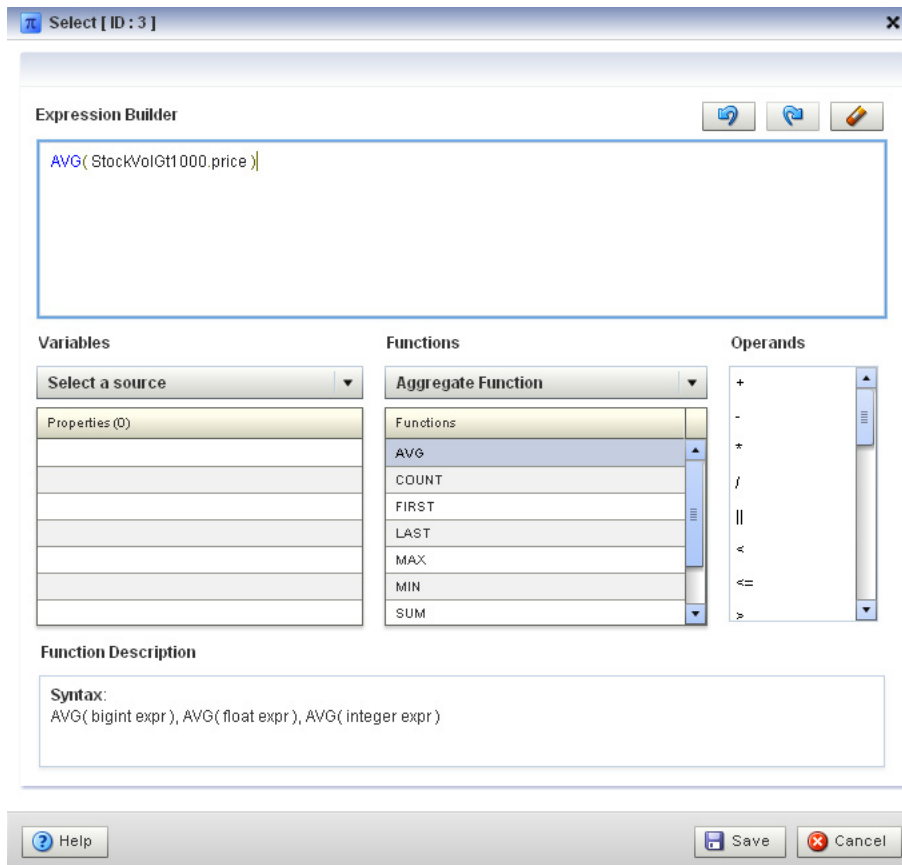
A list of the aggregate functions that Oracle CQL provides is displayed. You are going to use the **AVG** function.

32. Select the **StockVolGt1000.price** in the Expression Builder field.

33. Double-click the **AVG** function.

The `AVG()` function is wrapped around your selection in the Expression Builder field as [Figure 2–61](#) shows.

**Figure 2–61 Expression Builder: Applying the AVG Function**



**34. Click Save.**

The expression is added to the Project Expression field as [Figure 2–62](#) shows.



**Figure 2–62 Select Configuration Dialog: With Expression**

Step 1- Project

Distinct Results      Target Event Type: **MovingAvgEvent**

Source Properties (select from here): **StockVolGt1000**

| Properties (4) |                  |
|----------------|------------------|
| price          | double           |
| symbol         | java.lang.String |
| volume         | long             |
| ELEMENT_TIME   | timestamp        |

Selected Properties:

| Select List (1)              |
|------------------------------|
| StockVolGt1000.symbol:symbol |

Project Expression: `AVG( StockVolGt1000.price )`      AS: **AS**

Generated CQL Statement:

```
SELECT StockVolGt1000.symbol AS symbol from StockVolGt1000 [partition by symbol rows 2]
```

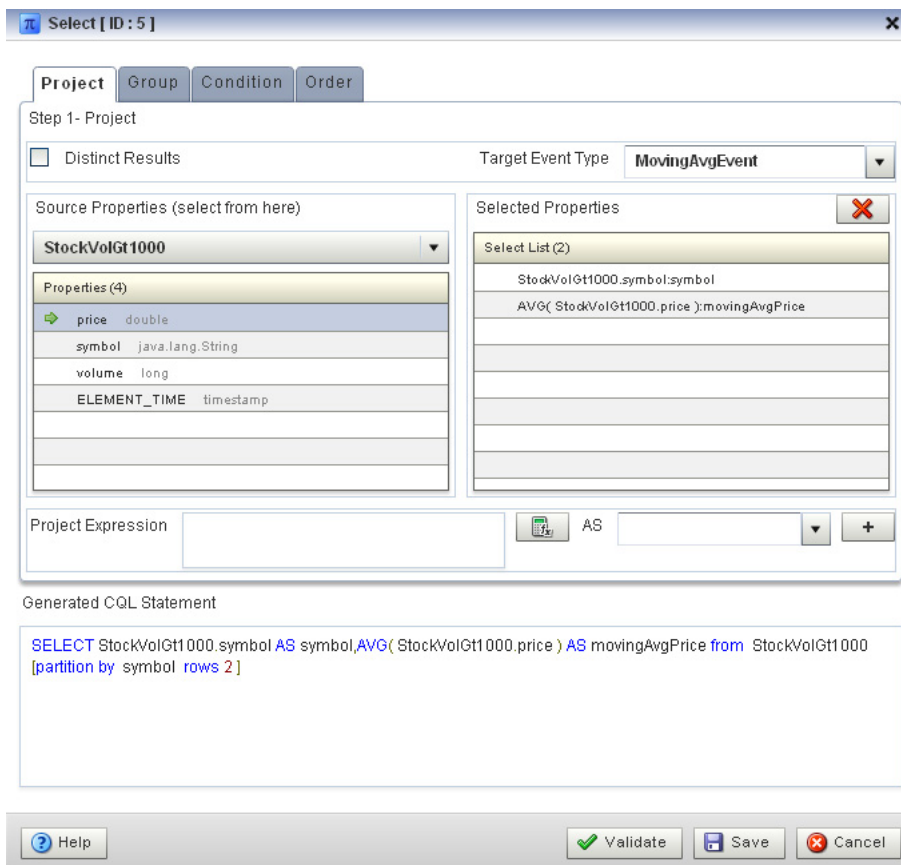
Buttons: Help, Validate, Save, Cancel

35. Click on the pull-down menu next to the AS field and select **movingAvgPrice**.

36. Click the plus Sign button.

The source property is added to the project expression of the Generated CQL Statement as [Figure 2–63](#) shows.

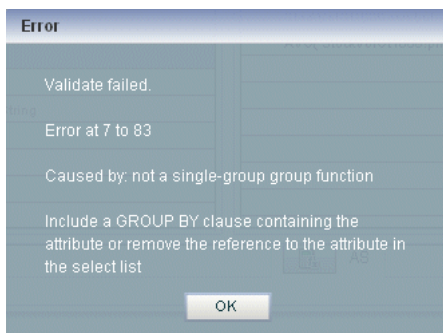
**Figure 2–63 Select Configuration Dialog: Source Property price Mapped to Output Event Property**



37. Click **Validate**.

A validation error dialog is shown as [Figure 2–64](#) shows.

**Figure 2–64 Validation Error: GROUP BY**



Because you are partitioning, you must specify a `GROUP BY` clause.

38. Select the **Group** tab.

The Group tab appears.

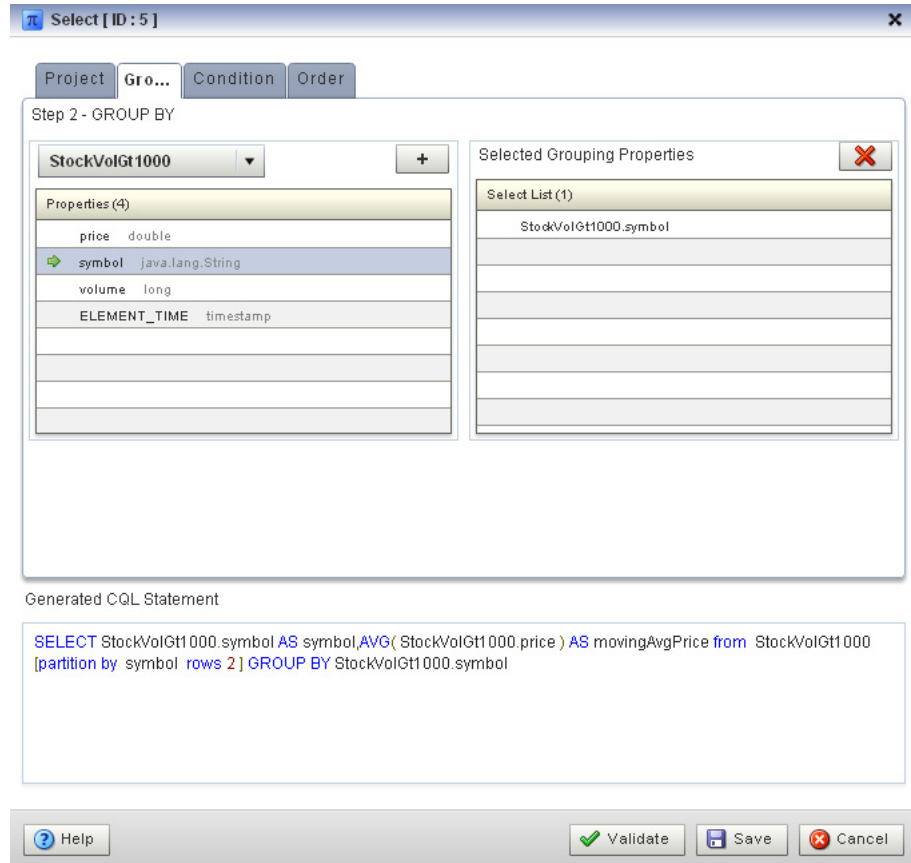
39. Configure the Group tab as follows (as shown in [Figure 2–65](#)):

- Select **StockVolGt1000** from the **Select a source** pull-down menu.

- Select **symbol** from the **Properties** list.
- Click the Plus Sign button.

The `symbol` property is added to `GROUP BY` clause as [Figure 2–65](#) shows.

**Figure 2–65 Group Tab: With symbol Grouping Property**



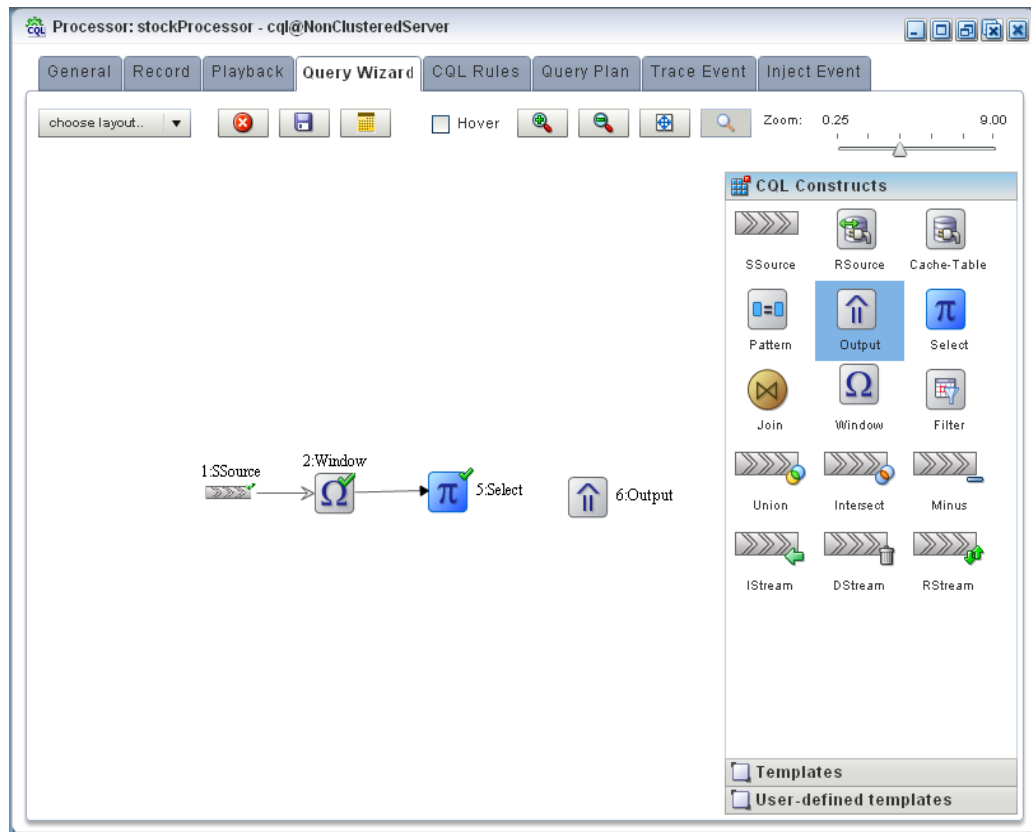
**40.** Click **Save**.

**41.** Click **Save Query**.

Next, you want to connect the query to an output.

**42.** Click and drag an Output icon from the CQL Constructs palette and drop it anywhere in the Query Wizard canvas as [Figure 2–66](#) shows.

**Figure 2–66 Query Wizard: Output**



43. Click on the **Select** icon and drag to the **Output** icon to connect the Oracle CQL constructs.
44. Double-click the **Output** icon.  
The Output configuration screen appears.
45. Configure the Output as follows (as shown in [Figure 2–67](#)):
  - Select **Query**.
  - Enter **MovingAverage** as the **Query Name**.

**Figure 2–67 Output Configuration Dialog**

The dialog shows the configuration for a query named "MovingAverage". The "Query" radio button is selected, and the "Enable" checkbox is checked. The "View" radio button is unselected. The "View Name" and "View Schema" fields are empty. The "Project List" table contains two entries:

| Properties (2) |                                            |
|----------------|--------------------------------------------|
| 1              | StockVolGt1000.symbol:symbol               |
| 2              | AVG( StockVolGt1000.price ):movingAvgPrice |
|                |                                            |
|                |                                            |
|                |                                            |

The "Generated CQL Statement" field contains the following SQL:

```
SELECT StockVolGt1000.symbol AS symbol,AVG( StockVolGt1000.price ) AS movingAvgPrice from StockVolGt1000
[partition by symbol rows 2] GROUP BY StockVolGt1000.symbol
```

The dialog has a toolbar at the bottom with buttons for Help, Inject Rule, Replace Rule, Validate, Save, and Cancel. A mouse cursor is pointing at the "Inject Rule" button.

**46. Click Inject Rule.**

The Inject Rule Confirmation dialog appears as [Figure 2–68](#) shows.

**Figure 2–68 Inject Rule Confirmation Dialog**

The dialog is titled "Confirmation" and contains the message "Rule has been successfully added." with an "OK" button below it.

**47. Click OK.**

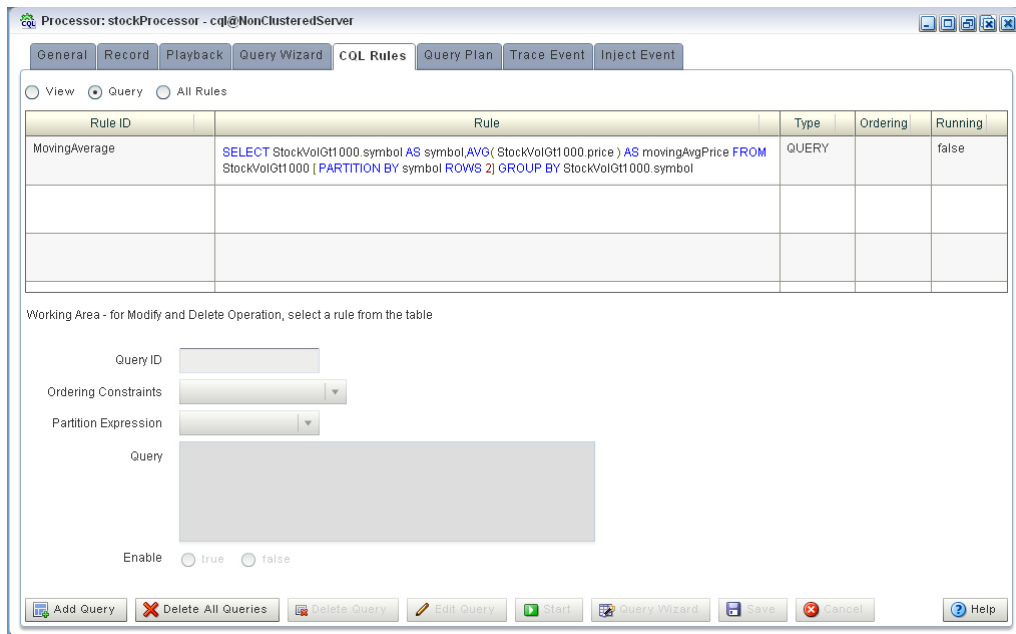
The Query Wizard adds the rule to the cqlProc processor.

**48. Click Save.****49. Click on the CQL Rules tab.**

The CQL Rules tab appears as [Figure 2–69](#) shows.

**50. Click on the Query radio button.**

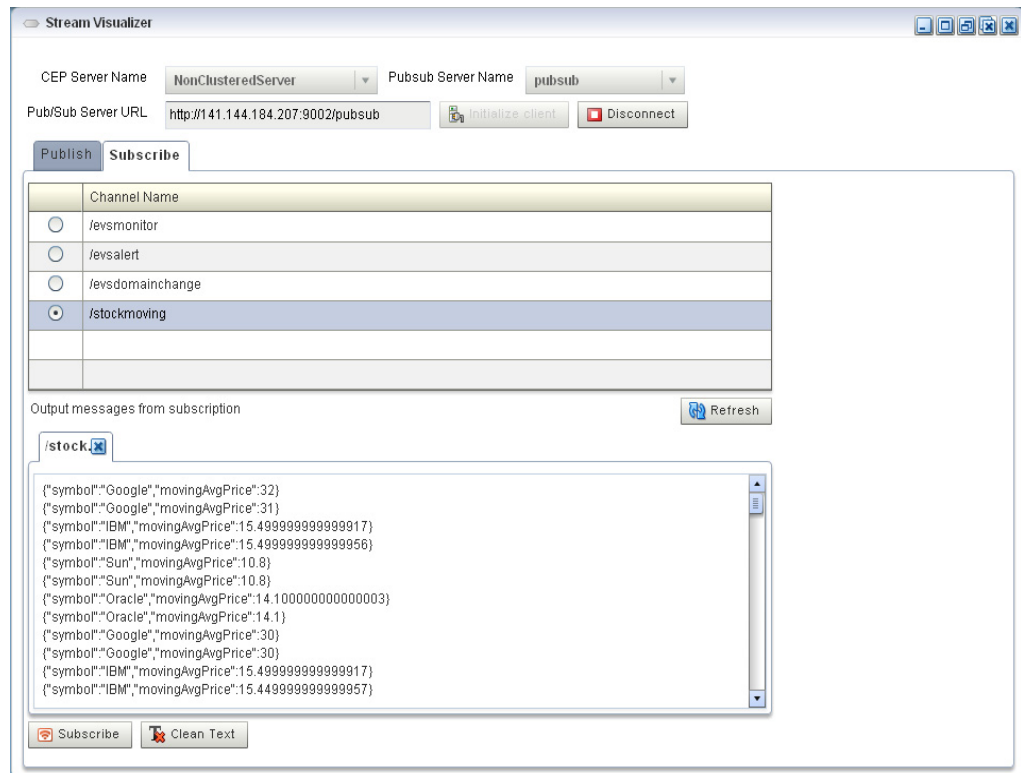
Confirm that your MovingAverage query is present.

**Figure 2–69 CQL Rules Tab With View MovingAverage****To test the moving average query:**

- To simulate the data feed for the moving average query, open a new command window and set your environment as described in [Section , "Setting Your Development Environment."](#)
- Change to the `MIDDLEWARE_HOME\ocep_11.1\utils\load-generator` directory, where `MIDDLEWARE_HOME` refers to the Middleware directory you specified when you installed Oracle Event Processing, such as `d:\Oracle\Middleware`.
- Run the load generator using the `stockData.prop` properties file:
  - On Windows:
 

```
prompt> runloadgen.cmd stockData.prop
```
  - On UNIX:
 

```
prompt> runloadgen.sh stockData.prop
```
- In the Oracle Event Processing Visualizer, click the **ViewStream** button in the top pane.  
The Stream Visualizer screen appears as [Figure 2–70](#) shows.

**Figure 2–70 Stream Visualizer: Showing Moving Average Query Output**

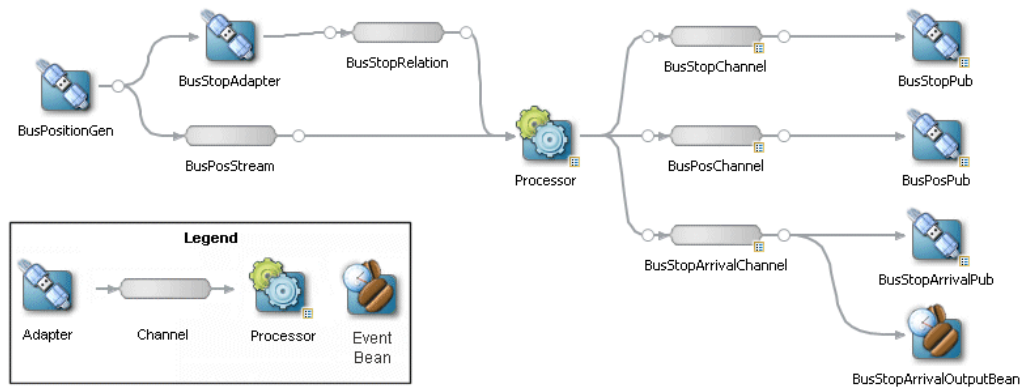
5. Click **Initialize Client**.
6. Enter `/stockmoving` in the **Initialize client** field.
7. Click **Subscribe**.

As the moving average query outputs events, the Oracle Event Processing updates the **Received Messages** area showing the events generated.

## Oracle Spatial Example

This example shows how to use Oracle Spatial with Oracle CQL queries to process a stream of Global Positioning System (GPS) events to track the GPS location of buses and generate alerts when a bus arrives at its pre-determined bus stop positions.

[Figure 2–71](#) shows Oracle Spatial example Event Processing Network (EPN). The EPN contains the components that make up the application and how they fit together.

**Figure 2–71 Oracle Spatial Example Event Processing Network**

The example includes the following components:

- **BusPositionGen**—Component that simulates an input stream of bus position GPS events. It uses the Oracle Event Processing loadgen utility and csvgen adapter provider to read in comma separated values (CSV) and deliver them to the EPN as BusPos events.
- **BusStopAdapter**—Custom adapter component that generates bus stop positions based on `MIDDLEWARE_HOME\ocep_11.1\samples\domains\spatial_domain\defaultserver\applications\spatial_sample\bus_stops.csv`, where `MIDDLEWARE_HOME` refers to the Middleware directory you specified when you installed Oracle Event Processing, such as `d:\Oracle\Middleware`.
- **BusPosStream**—Component that transmits BusPos events to the Processor as a stream.
- **BusStopRelation**—Component that transmits BusPos events to the Processor as a relation.
- **Processor**—Component that executes Oracle CQL queries on the incoming BusPos events.
- **BusStopChannel, BusPosChannel, and BusStopArrivalChannel**—Components that each specify a different selector to transmit the results of a different query from the Processor component to the appropriate outbound adapter or output bean.
- **BusStopPub, BusPosPub, and BusStopArrivalPub**—Components that publish the results of the Processor component's queries.
- **BusStopArrivalOutputBean**—POJO event bean component that logs a message for each insert, delete, and update event to help visualize the relation offered by the BusStopArrivalChannel.

---

**Note:** For more information about data cartridges, see:

- "Introduction to Data Cartridges" in the *Oracle Fusion Middleware CQL Language Reference for Oracle Event Processing*
  - "Oracle Spatial" in the *Oracle Fusion Middleware CQL Language Reference for Oracle Event Processing*
- 

## Running the Oracle Spatial Example

The Oracle Spatial application is pre-deployed to the `spatial_domain` domain. To run the application, you simply start an instance of Oracle Event Processing server.



**To run the Oracle Spatial example from the spatial\_domain domain:**

1. Open a command window and change to the default server directory of the Oracle Spatial example domain directory, located in `MIDDLEWARE_HOME\ocp_11.1\samples\domains\spatial_domain\defaultserver`, where `MIDDLEWARE_HOME` refers to the Middleware directory you specified when you installed Oracle Event Processing, such as `d:\Oracle\Middleware`.

```
prompt> cd d:\Oracle iddleware\ocp_11.1\samples\domains\spatial_
domain\defaultserver
```

2. Ensure the environment is set correctly in the server startup script.

For more information, see [Chapter 3, "Getting Started with Developing Oracle Event Processing Applications"](#)

3. Start Oracle Event Processing by executing the appropriate script with the correct command line arguments:

- a. On Windows:

- \* If you are using the JRockit JDK included in Oracle JRockit Real Time, enable the deterministic garbage collector by passing the `-dgc` parameter to the command:

```
prompt> startwlevs.cmd -dgc
```

- \* If you are not using the JRockit JDK included in Oracle JRockit Real Time:

```
prompt> startwlevs.cmd
```

- b. On UNIX:

- \* If you are using the JRockit JDK included in Oracle JRockit Real Time, enable the deterministic garbage collector by passing the `-dgc` parameter to the command:

```
prompt> startwlevs.sh -dgc
```

- \* If you are not using the JRockit JDK included in Oracle JRockit Real Time:

```
prompt> startwlevs.sh
```

Wait for the console log to show:

```
<Mar 4, 2010 2:13:15 PM EST> <Notice> <Spring> <BEA-2047000> <The application
context for "spatial_sample" was started successfully>
<Mar 4, 2010 2:13:15 PM EST> <Notice> <Server> <BEA-2046000> <Server STARTED>
```

This message indicates that the Oracle Spatial example is running correctly.

4. On the same host as the Oracle Spatial example is running, launch a browser and navigate to `http://localhost:9002/bus/main.html`.

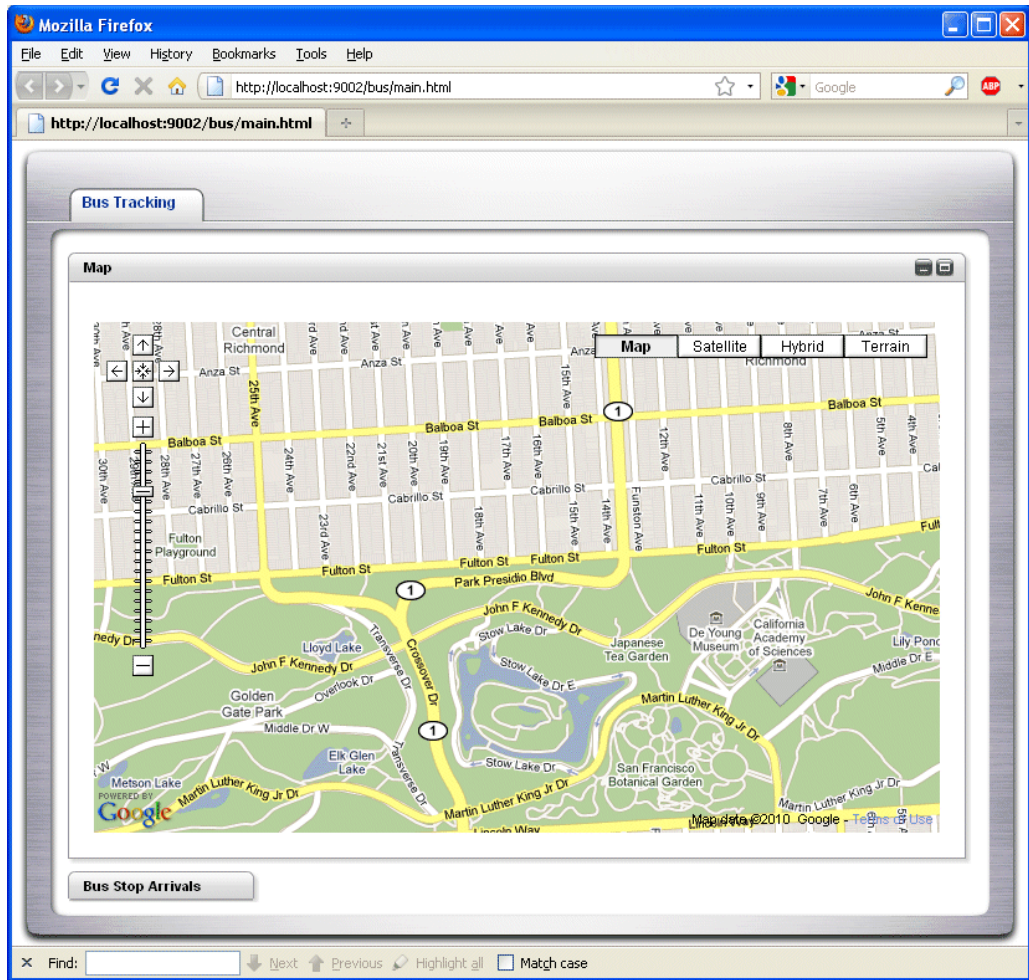
---

**Note:** You cannot run this example on one host and browse to it from another host. This is a limitation of the Google API Key that the example uses and is not a limitation of Oracle Event Processing.

---

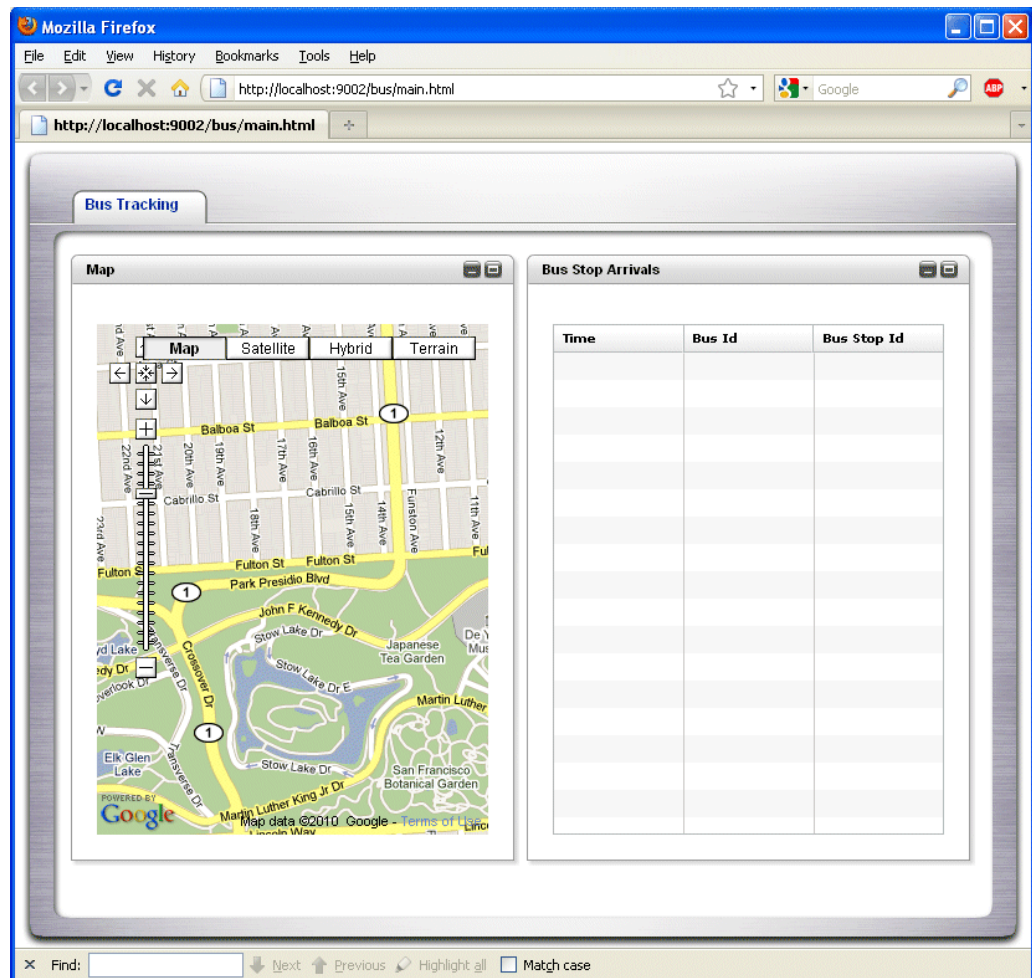
The Oracle Spatial example Web page appears as [Figure 2-72](#) shows.

Figure 2–72 Oracle Spatial Web Page



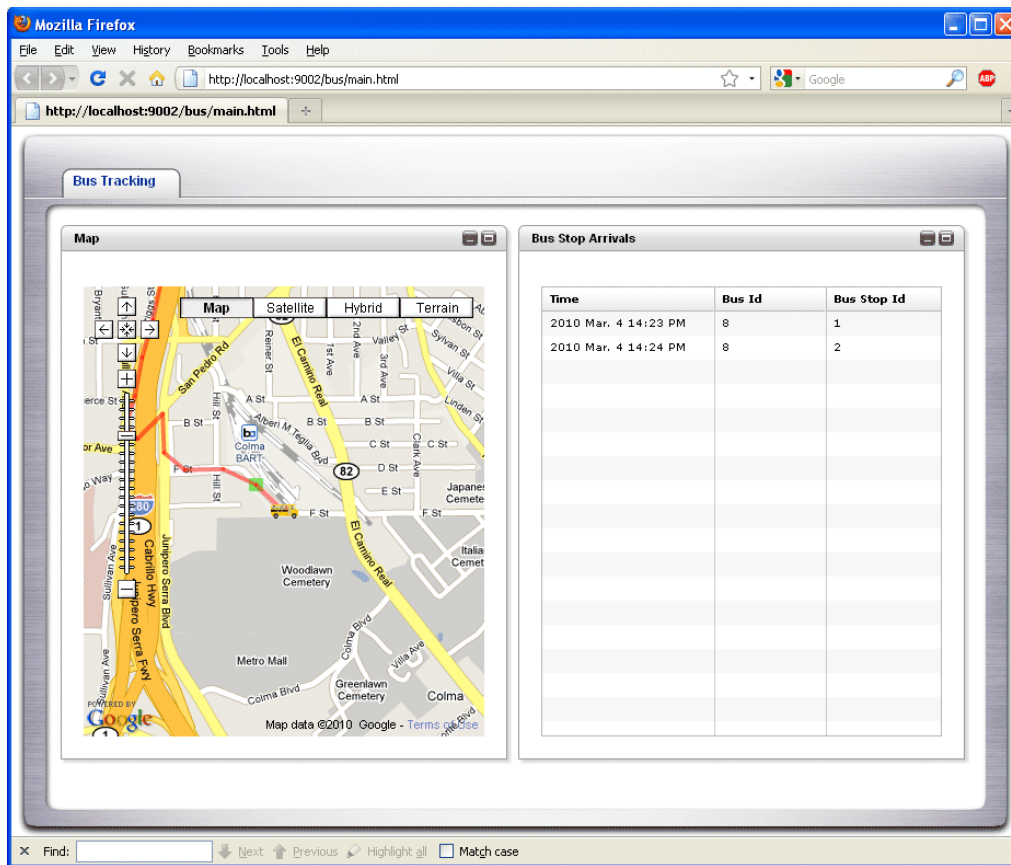
Click the **Bus Top Arrivals** tab to view bus stop arrivals as [Figure 2–73](#) shows.

Figure 2–73 Oracle Spatial Web Page: Bus Stop Arrivals Tab



5. Execute the Oracle Event Processing load generator to generate sample data:
  - a. On Windows:
    - \* Open a command prompt and navigate to `MIDDLEWARE_HOME/occep_11.1/utills/load-generator`
    - \* `runloadgen.cmd bus_positions.prop`
  - b. On UNIX:
    - \* Open a terminal window and navigate to `MIDDLEWARE_HOME/occep_11.1/utills/load-generator`
    - \* `runloadgen.sh bus_positions.prop`
6. Observe the bus movements and alerts in the browser as [Figure 2–74](#) shows.

Figure 2-74 Oracle Spatial Web Page: Bus Tracking



## Building and Deploying the Oracle Spatial Example

The Oracle Spatial sample source directory contains the Java source, along with other required resources such as configuration XML files, that make up the Oracle Spatial application. The `build.xml` Ant file contains targets to build and deploy the application to the `spatial_domain` domain.

For more information, see [Section , "Description of the Ant Targets to Build Hello World"](#).

### To build and deploy the Oracle Spatial example from the source directory:

1. If the `spatial_domain` Oracle Event Processing instance is not already running, follow the procedure in [Section , "Running the Oracle Spatial Example"](#) to start the server.

You must have a running server to successfully deploy the rebuilt application.

2. Open a new command window and change to the Oracle Spatial source directory, located in `MIDDLEWARE_HOME\ocep_11.1\samples\source\applications\spatial` where `MIDDLEWARE_HOME` is the Middleware directory you specified when you installed Oracle Event Processing.

For example:

```
prompt> cd d:\Oracle\Middleware\ocep_11.1\samples\source\applications\spatial
```

3. Set your development environment as described in [Section , "Setting Your Development Environment."](#)

4. Execute the `all` Ant target to compile and create the application JAR file:

```
prompt> ant all
```

5. Execute the `deploy` Ant target to deploy the application JAR file to Oracle Event Processing:

```
prompt> ant -Daction=update deploy
```

---

**Caution:** This target overwrites the existing Oracle Spatial application JAR file in the domain directory.

---

## Description of the Ant Targets to Build the Oracle Spatial Example

The `build.xml` file, located in the top level of the Oracle Spatial source directory, contains the following targets to build and deploy the application:

- `clean`—This target removes the `dist` and output working directories under the current directory.
- `all`—This target cleans, compiles, and JARs up the application into a file called `com.bea.wlevs.example.helloworld_11.1.1.4_0.jar`, and places the generated JAR file into a `dist` directory below the current directory.
- `deploy`—This target deploys the JAR file to Oracle Event Processing using the Deployer utility.

For more information, see "Deployer Command-Line Reference" in the *Oracle Fusion Middleware Administrator's Guide for Oracle Event Processing*.

## Implementation of the Oracle Spatial Example

The implementation of the Oracle Spatial example generally follows "Creating Oracle Event Processing Applications: Typical Steps" in the *Oracle Fusion Middleware Developer's Guide for Oracle Event Processing for Eclipse*.

Refer to that section for a task-oriented procedure that describes the typical development process.

All the files of the Oracle Spatial example are located relative to the `MIDDLEWARE_HOME\ocep_11.1\samples\source\applications\spatial` directory, where `MIDDLEWARE_HOME` refers to the Middleware directory you specified when you installed Oracle Event Processing, such as `c:\Oracle\Middleware`. Oracle recommends that you use this example directory setup in your own environment, although it is obviously not required.

The files used by the Oracle Spatial example include:

- An EPN assembly file that describes each component in the application and how all the components are connected together. You are required to include this XML file in your Oracle Event Processing application.

In the example, the file is called `context.xml` and is located in the `META-INF/spring` directory.

- A component configuration file that configures the various components on the EPN including the processor component of the application:

In the example, this file is called `config.xml` and is located in the `META-INF/wlevs` directory.

- Java files that implement:
  - `BusStopAdapter`: Custom adapter component that generates bus stop positions based on `MIDDLEWARE_HOME\ocep_11.1\samples\domains\spatial_domain\defaultserver\applications\spatial_sample\bus_stops.csv`, where `MIDDLEWARE_HOME` refers to the Middleware directory you specified when you installed Oracle Event Processing, such as `d:\Oracle\Middleware`.
  - `OutputBean`: POJO event bean component that logs a message for each insert, delete, and update event to help visualize the relation offered by the `BusStopArrivalChannel`
  - `OrdsHelper`: Helper class that provides method `getOrds` to return the ordinates from a `JGeometry` as a `List of Double` values.

These Java files are located in the `source\applications\spatial\src\com\oracle\cep\sample\spatial` directory.

For additional information about the Oracle Event Processing APIs referenced in this POJO, see *Oracle Fusion Middleware Java API Reference for Oracle Event Processing*.

- A `MANIFEST.MF` file that describes the contents of the OSGi bundle that will be deployed to Oracle Event Processing.

In the example, the `MANIFEST.MF` file is located in the `META-INF` directory.

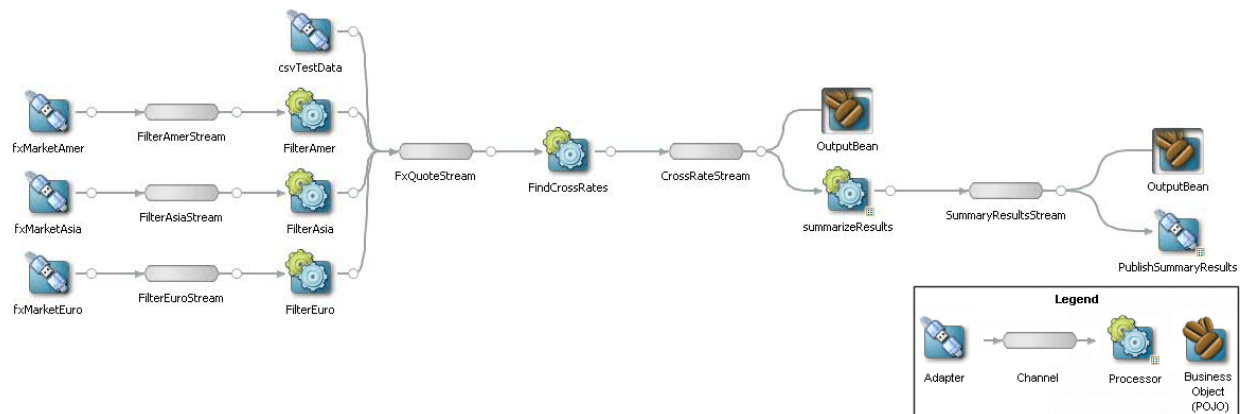
For more information about creating this file, as well as a description of creating the OSGi bundle that you deploy to Oracle Event Processing, see [Section , "Overview of Application Assembly and Deployment"](#).

The Oracle Spatial example uses a `build.xml` Ant file to compile, assemble, and deploy the OSGi bundle; see [Section , "Building and Deploying the Oracle Spatial Example"](#) for a description of this `build.xml` file if you also use Ant in your development environment.

## Foreign Exchange (FX) Example

The foreign exchange example, called FX for simplicity, is a more complex example than the HelloWorld example because it includes multiple processors that handle information from multiple data feeds. In the example, the data feeds are simulated using the Oracle Event Processing load generator utility.

[Figure 2-75](#) shows the FX example Event Processing Network (EPN). The EPN contains the components that make up the application and how they fit together.

**Figure 2–75 FX Example Event Processing Network**

In this scenario, three data feeds, simulated using the load generator, send a constant pair of values from different parts of the world; the value pairs consist of a currency pair, such as USDEUR for US dollar - European euro, and an exchange rate between the two currencies. The `fxMarketAmer`, `fxMarketAsia`, and `fxMarketEuro` adapters receive the data from the feeds, convert them into events, and pass them to the corresponding `FilterAmer`, `FilterAsia`, and `FilterEuro` processors. Each processor performs an initial stale check to ensure that no event is more than 1 second old and then a boundary check to ensure that the exchange rate between the two currencies is within a current boundary. The processor also only selects a specific currency pair from a particular channel; for example, the server selects USDEUR from the simulated American data feed, but rejects all other pairs, such as USDAUD (Australian dollar).

After the data from each data feed provider passes this initial preparation phase, a different processor, called `FindCrossRate`, joins all events across all providers, calculates the mid-point between the maximum and minimum rate, and then applies a trader-specified spread. Finally, the processor forwards the rate to the POJO that contains the business code; in this example, the POJO simply publishes the rate to clients.

The Oracle Event Processing monitor is configured to watch if the event latency in the last step exceeds some threshold, such as no updated rates in a 30 second time-span, and if there is too much variance between two consecutive rates for the same currency pair. Finally, the last rate of each currency pair is forwarded to the Oracle Event Processing http pub-sub server.

## Running the Foreign Exchange Example

For optimal demonstration purposes, Oracle recommends that you run this example on a powerful computer, such as one with multiple CPUs or a 3 GHz dual-core Intel, with a minimum of 2 GB of RAM.

The Foreign Exchange (FX) application is pre-deployed to the `fx_domain` domain. To run the application, you simply start an instance of Oracle Event Processing server.

### To run the foreign exchange example:

1. Open a command window and change to the default server directory of the FX domain directory, located in `MIDDLEWARE_HOME\ocep_11.1\samples\domains\fx_domain\defaultserver`, where `MIDDLEWARE_HOME` refers to the Middleware directory you specified when you installed Oracle Event Processing, such as `d:\Oracle\Middleware`.

```
prompt> cd d:\Oracle\middleware\ocep_11.1\samples\domains\fx_
domain\defaultserver
```

2. Set your development environment, as described in [Section 3, "Setting Your Development Environment."](#)
3. Start Oracle Event Processing by executing the appropriate script with the correct command line arguments:

**a. On Windows:**

- \* If you are using the JRockit JDK included in Oracle JRockit Real Time, enable the deterministic garbage collector by passing the `-dgc` parameter to the command:

```
prompt> startwlevs.cmd -dgc
```

- \* If you are not using the JRockit JDK included in Oracle JRockit Real Time:

```
prompt> startwlevs.cmd
```

**b. On UNIX:**

- \* If you are using the JRockit JDK included in Oracle JRockit Real Time, enable the deterministic garbage collector by passing the `-dgc` parameter to the command:

```
prompt> startwlevs.sh -dgc
```

- \* If you are not using the JRockit JDK included in Oracle JRockit Real Time:

```
prompt> startwlevs.sh
```

The FX application is now ready to receive data from the data feeds.

4. To simulate an American data feed, open a new command window and set your environment as described in [Section 3, "Getting Started with Developing Oracle Event Processing Applications."](#)
5. Change to the `MIDDLEWARE_HOME\ocep_11.1\utils\load-generator` directory, where `MIDDLEWARE_HOME` refers to the Middleware directory you specified when you installed Oracle Event Processing, such as `d:\Oracle\Middleware`.
6. Run the load generator using the `fxAmer.prop` properties file:

**a. On Windows:**

```
prompt> runloadgen.cmd fxAmer.prop
```

**b. On UNIX:**

```
prompt> runloadgen.sh fxAmer.prop
```

7. Repeat steps 4 - 6 to simulate an Asian data feed, using the `fxAsia.prop` properties file:

**a. On Windows:**

```
prompt> runloadgen.cmd fxAsia.prop
```

**b. On UNIX:**

```
prompt> runloadgen.sh fxAsia.prop
```



8. Repeat steps 4 - 6 to simulate an European data feed, using the `fxEuro.prop` properties file:

- a. On Windows:

```
prompt> runloadgen.cmd fxEuro.prop
```

- b. On UNIX:

```
prompt> runloadgen.sh fxEuro.prop
```

After the server status messages scroll by in the command window from which you started the server, and the three load generators start, you should see messages similar to the following being printed to the server command window (the message will likely be on one line):

```
OutputBean:onEvent() +
  <TupleValue>
    <EventType>SpreaderOutputEvent</EventType>
    <ObjectName>FindCrossRatesRule</ObjectName>
    <Timestamp>1843704855846</Timestamp>
    <TupleKind>null</TupleKind>
    <DoubleAttribute>
      <Value>90.08350000074516</Value>
    </DoubleAttribute>
    <CharAttribute>
      <Value>USD</Value>
      <Length>3</Length>
    </CharAttribute>
    <CharAttribute>
      <Value>JPY</Value>
      <Length>3</Length>
    </CharAttribute>
    <IsTotalOrderGuarantee>>false</IsTotalOrderGuarantee>
  </TupleValue>
```

These messages indicate that the Foreign Exchange example is running correctly. The output shows the cross rates of US dollars to Japanese yen and US dollars to UK pounds sterling.

## Building and Deploying the Foreign Exchange Example from the Source Directory

The Foreign Exchange (FX) sample source directory contains the Java source, along with other required resources such as configuration XML files, that make up the FX application. The `build.xml` Ant file contains targets to build and deploy the application to the `fx_domain` domain, as described in [Section , "Description of the Ant Targets to Build Hello World."](#)

### To build and deploy the foreign exchange example from the source directory:

1. If the FX Oracle Event Processing instance is not already running, follow the procedure in [Section , "Running the Foreign Exchange Example"](#) to start the server.

You must have a running server to successfully deploy the rebuilt application.

2. Open a new command window and change to the FX source directory, located in `MIDDLEWARE_HOME\ocep_11.1\samples\source\applications\fx`, where `MIDDLEWARE_HOME` refers to the Middleware directory you specified when you installed Oracle Event Processing installation directory, such as `d:\Oracle\Middleware`.

```
prompt> cd d:\Oracle\Middleware\ocep_11.1\samples\source\applications\fx
```

3. Set your development environment, as described in [Section , "Setting Your Development Environment."](#)
4. Execute the all Ant target to compile and create the application JAR file:  

```
prompt> ant all
```
5. Execute the deploy Ant target to deploy the application JAR file to Oracle Event Processing:

```
prompt> ant -Dusername=wlevs -Dpassword=wlevs -Daction=update deploy
```

---

---

**Caution:** This target overwrites the existing FX application JAR file in the domain directory.

---

---

6. If the load generators required by the FX application are not running, start them as described in [Section , "Running the Foreign Exchange Example."](#)

After server status messages scroll by, you should see the following message printed to the output:

```
{crossRate=USDJPY, internalPrice=119.09934499999781}, {crossRate=USDGBP,  
internalPrice=0.5031949999999915}, {crossRate=USDJPY,  
internalPrice=117.73945624999783}
```

This message indicates that the FX example has been redeployed and is running correctly.

## Description of the Ant Targets to Build FX

The `build.xml` file, located in the top-level directory of the FX source, contains the following targets to build and deploy the application:

- `clean`—This target removes the `dist` and output working directories under the current directory.
- `all`—This target cleans, compiles, and jars up the application into a file called `com.bea.wlevs.example.fx_11.1.1.4_0.jar`, and places the generated JAR file into a `dist` directory below the current directory.
- `deploy`—This target deploys the JAR file to Oracle Event Processing using the Deployer utility.

For more information, see "Deployer Command-Line Reference" in the *Oracle Fusion Middleware Administrator's Guide for Oracle Event Processing*.

## Implementation of the FX Example

The implementation of the foreign exchange (FX) example generally follows "Creating Oracle Event Processing Applications: Typical Steps" in the *Oracle Fusion Middleware Developer's Guide for Oracle Event Processing for Eclipse*.

Refer to that section for a task-oriented procedure that describes the typical development process.

All the files of the FX example are located relative to the `MIDDLEWARE_HOME\ocep_11.1\samples\source\applications\fx` directory, where `MIDDLEWARE_HOME` is the

Middleware home directory you specified when you installed Oracle Event Processing `c:\Oracle\Middleware`. Oracle recommends that you use this example directory setup in your own environment, although it is obviously not required.

The files used by the FX example include:

- An EPN assembly file that describes each component in the application and how all the components are connected together. You are required to include this XML file in your Oracle Event Processing application.

In the example, the file is called `com.oracle.cep.sample.fx.context.xml` and is located in the `META-INF/spring` directory.

- Two XML files that configure the processor components of the application:

The first XML file configures the `filterAmer`, `filterAsia`, `filterEuro`, and `FindCrossRates` processors, all in a single file. This XML file includes the Oracle CQL rules that select particular currency pairs from particular simulated market feeds and joins together all the events that were selected by the pre-processors, calculates an internal price for the particular currency pair, and then calculates the cross rate. In the example, this file is called `spreader.xml` and is located in the `META-INF/wlevs` directory.

The second XML file configures the `summarizeResults` processor and includes the Oracle CQL rule that summarizes the results of the `FindCrossRates` processor. In the example, this file is called `SummarizeResults.xml` and is located in the `META-INF/wlevs` directory.

- An XML file that configures the `PublishSummaryResults` `http` pub-sub adapter. In the example, this file is called `PubSubAdapterConfiguration.xml` and is located in the `META-INF/wlevs` directory.
- A Java file that implements the `OutputBean` component of the application, a POJO that contains the business logic. This POJO prints out to the screen the events that it receives, programmed in the `onEvent` method. The POJO also registers into the event type repository the `ForeignExchangeEvent` event type.

In the example, the file is called `OutputBean.java` and is located in the `src/com/oracle/cep/sample/fx` directory.

For additional information about the Oracle Event Processing APIs referenced in this POJO, see *Oracle Fusion Middleware Java API Reference for Oracle Event Processing*.

- A `MANIFEST.MF` file that describes the contents of the OSGi bundle that will be deployed to Oracle Event Processing.

In the example, the `MANIFEST.MF` file is located in the `META-INF` directory.

For more information about creating this file, as well as a description of creating the OSGi bundle that you deploy to Oracle Event Processing, see [Section , "Overview of Application Assembly and Deployment"](#).

The FX example uses a `build.xml` Ant file to compile, assemble, and deploy the OSGi bundle; see [Section , "Building and Deploying the Foreign Exchange Example from the Source Directory"](#) for a description of this `build.xml` file if you also use Ant in your development environment.

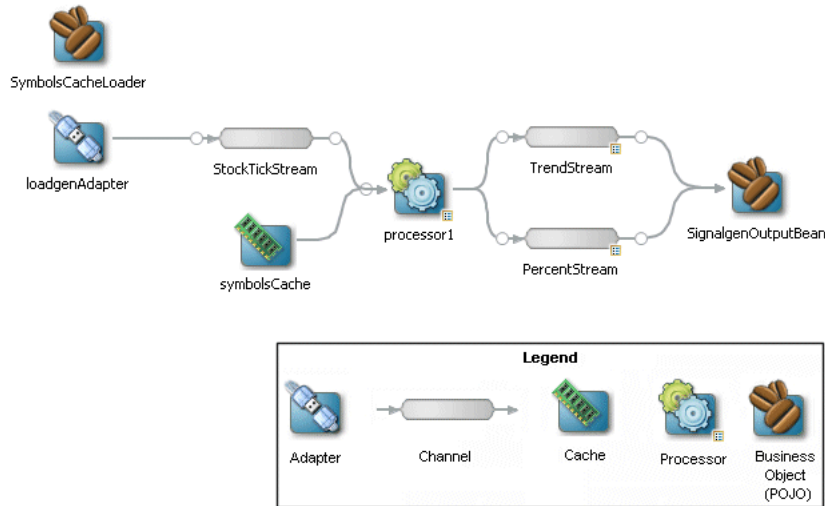
## Signal Generation Example

The signal generation sample application receives simulated market data and verifies if the price of a security has fluctuated more than two percent. The application also

detects the pattern occurring by keeping track of successive stock prices for a particular symbol; if more than three successive prices are larger than the one before it, this is considered a pattern.

Figure 2–76 shows the signal generation example Event Processing Network (EPN). The EPN contains the components that make up the application and how they fit together.

**Figure 2–76 The Signal Generation Example Event Processing Network**



The application simulates a market data feed using the Oracle Event Processing load generator utility; in this example, the load generator generates up to 10,000 messages per second. The example includes an HTML dashboard which displays the matched events along with the latencies; events consist of a stock symbol, a timestamp, and the price.

The example demonstrates very low latencies, with minimum latency *jitter* under high throughputs. Once the application starts running, the processor matches an average of 800 messages per second. If the application is run on the minimum configured system, the example shows very low average latencies (30-300 microsecond, on average) with minimal latency spikes (low milliseconds).

The example computes and displays latency values based on the difference between a timestamp generated on the load generator and timestamp on Oracle Event Processing. Computing valid latencies requires very tight clock synchronization, such as 1 millisecond, between the computer running the load generator and the computer running Oracle Event Processing. For this reason, Oracle recommends running both the load generator and Oracle Event Processing on a single multi-CPU computer where they will share a common clock.

The example also shows how to use the Oracle Event Processing event caching feature. In particular the single processor in the EPN sends events to both an event bean and a cache.

The example also demonstrates how to use Oracle CQL queries.

## Running the Signal Generation Example

For optimal demonstration purposes, Oracle recommends that you run this example on a powerful computer, such as one with multiple CPUs or a 3 GHz dual-core Intel, with a minimum of 2 GB of RAM.

The `signalgeneration_domain` domain contains a single application: the signal generation sample application. To run the signal generation application, you simply start an instance of Oracle Event Processing in that domain.

**To run the signal generation example:**

1. Open a command window and change to the default server directory of the `signalgeneration_domain` domain directory, located in `MIDDLEWARE_HOME\ocep_11.1\samples\domains\signalgeneration_domain\defaultserver`, where `MIDDLEWARE_HOME` refers to the Middleware home directory you specified when you installed Oracle Event Processing, such as `d:\Oracle\Middleware`.

```
prompt> cd d:\Oracle\Middleware\ocep_11.1\samples\domains\signalgeneration_
domain\defaultserver
```

2. Set your development environment, as described in [Section , "Setting Your Development Environment."](#)
3. Start Oracle Event Processing by executing the appropriate script with the correct command line arguments:

**a. On Windows:**

- \* If you are using the JRockit JDK included in Oracle JRockit Real Time, enable the deterministic garbage collector by passing the `-dgc` parameter to the command:

```
prompt> startwlevs.cmd -dgc
```

- \* If you are not using the JRockit JDK included in Oracle JRockit Real Time:

```
prompt> startwlevs.cmd
```

**b. On UNIX:**

- \* If you are using the JRockit JDK included in Oracle JRockit Real Time, enable the deterministic garbage collector by passing the `-dgc` parameter to the command:

```
prompt> startwlevs.sh -dgc
```

- \* If you are not using the JRockit JDK included in Oracle JRockit Real Time:

```
prompt> startwlevs.sh
```

4. Wait until you see console messages like this:

```
<Apr 24, 2009 11:40:37 AM EDT> <Notice> <Server> <BEA-2046000> <Server STARTED>
Throughput (msg per second): 0. Average latency (microseconds): 0
Throughput (msg per second): 0. Average latency (microseconds): 0
Throughput (msg per second): 0. Average latency (microseconds): 0
Throughput (msg per second): 0. Average latency (microseconds): 0
...
```

The signal generation application is now ready to receive data from the data feeds.

Next, to simulate a data feed, you use a load generator programmed specifically for the example.

5. Open a new command window.
6. Change to the `MIDDLEWARE_HOME\ocep_11.1\samples\domains\signalgeneration_domain\defaultserver\utils` directory, where `MIDDLEWARE_HOME` refers to the Middleware home directory you

specified when you installed Oracle Event Processing, such as `d:\Oracle\Middleware`.

7. Run the `startDataFeed` command:

a. On Windows:

```
prompt> startDataFeed.cmd
```

b. On UNIX:

```
prompt> startDataFeed.sh
```

8. Invoke the example dashboard by starting a browser and opening the following HTML page:

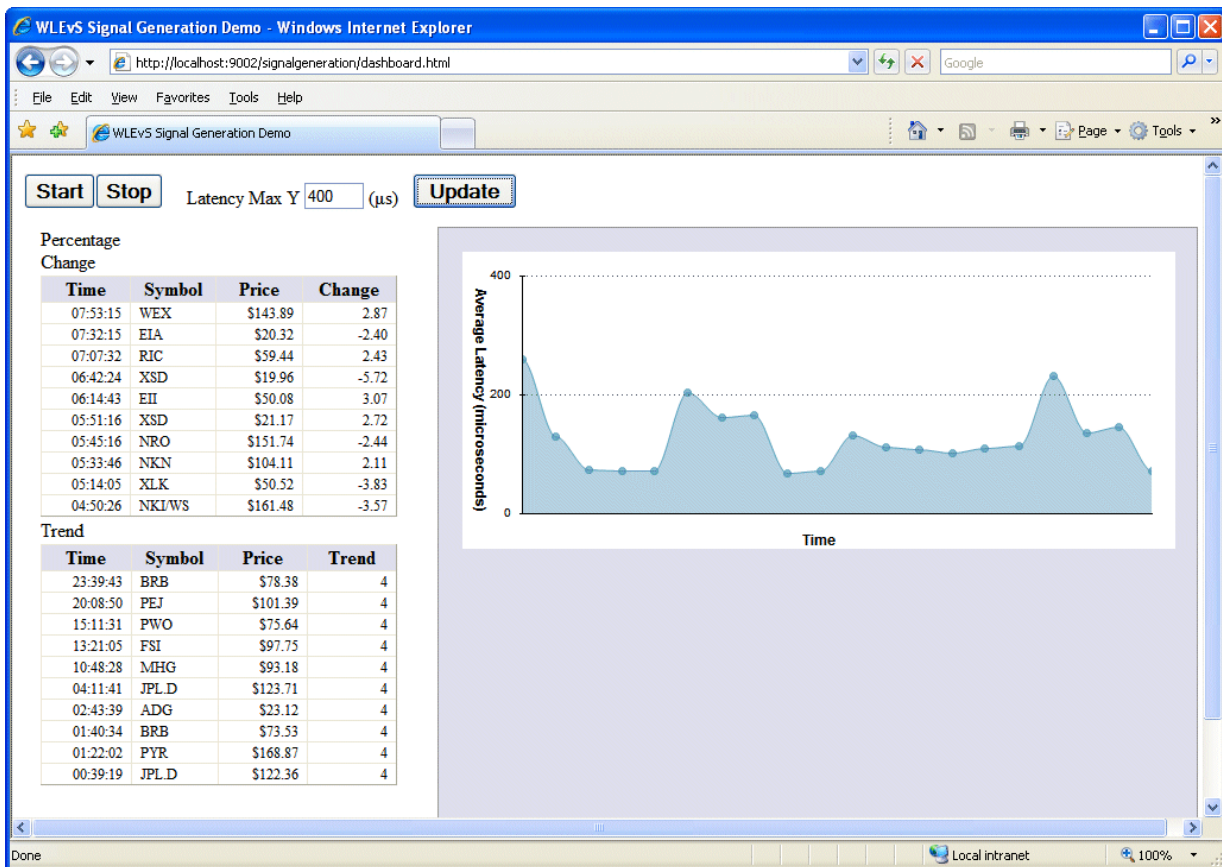
```
http://host:9002/signalgeneration/dashboard.html
```

Replace `host` with the name of the computer on which Oracle Event Processing is running; if it is the same computer as your browser, you can use `localhost`.

9. In the browser, click **Start** on the HTML page.

You should start seeing the events that match the Oracle CQL rules configured for this example as [Figure 2-77](#) shows.

**Figure 2-77** Signal Generation Dashboard



## Building and Deploying the Signal Generation Example from the Source Directory

The signal generation sample source directory contains the Java source, along with other required resources, such as configuration XML files, EPN assembly file, and DOJO client JavaScript libraries, that make up the signal generation application. The `build.xml` Ant file contains targets to build and deploy the application to the `signalgeneration_domain` domain, as described in [Section , "Description of the Ant Targets to Build Signal Generation."](#)

### To build and deploy the signal generation example from the source directory:

1. If the signal generation Oracle Event Processing instance is not already running, follow the procedure in [Section , "Running the Signal Generation Example"](#) to start the server. You must have a running server to successfully deploy the rebuilt application.

2. Open a new command window and change to the signal generation source directory, located in `MIDDLEWARE_HOME\ocep_11.1\samples\source\applications\signalgeneration`, where `MIDDLEWARE_HOME` refers to the Middleware home directory you specified when you installed Oracle Event Processing, such as `d:\Oracle\Middleware`.

```
prompt> cd d:\Oracle\Middleware\ocep_
11.1\samples\source\applications\signalgeneration
```

3. Set your development environment, as described in [Section , "Setting Your Development Environment."](#)

4. Execute the `all` Ant target to compile and create the application JAR file:

```
prompt> ant all
```

5. Execute the `deploy` Ant target to deploy the application JAR file to the `MIDDLEWARE_HOME\ocep_11.1\samples\domains\signalgeneration_domain\defaultserver\applications\signalgeneration` directory:

```
prompt> ant deploy
```

---



---

**Caution:** This target overwrites the existing signal generation application JAR file in the domain directory.

---



---

6. If the load generator required by the signal generation application is not running, start it as described in [Section , "Running the Signal Generation Example."](#)
7. Invoke the example dashboard as described in [Section , "Running the Signal Generation Example."](#)

## Description of the Ant Targets to Build Signal Generation

The `build.xml` file, located in the top-level directory of the signal generation example source, contains the following targets to build and deploy the application:

- `clean`—This target removes the `dist` and output working directories under the current directory.
- `all`—This target cleans, compiles, and jars up the application into a file called `com.bea.wlevs.example.signalgen_11.1.1.4_0.jar`, and places the generated JAR file into a `dist` directory below the current directory.

- `deploy`—This target deploys the JAR file to Oracle Event Processing using the Deployer utility.

For more information, see "Deployer Command-Line Reference" in the *Oracle Fusion Middleware Administrator's Guide for Oracle Event Processing*.

## Implementation of the Signal Generation Example

The implementation of the signal generation example generally follows "Creating Oracle Event Processing Applications: Typical Steps" in the *Oracle Fusion Middleware Developer's Guide for Oracle Event Processing for Eclipse*.

Refer to that section for a task-oriented procedure that describes the typical development process.

All the files of the signal generation are located relative to the `MIDDLEWARE_HOME\ocep_11.1\samples\source\applications\signalgeneration` directory, where `MIDDLEWARE_HOME` refers to the Middleware home directory you specified when you installed Oracle Event Processing, such as `c:\Oracle\Middleware`. Oracle recommends that you use this example directory setup in your own environment, although it is obviously not required.

The files used by the signal generation example include:

- A EPN assembly file that describes each component in the application and how all the components are connected together.

In the example, the file is called `epn_assembly.xml` and is located in the `META-INF/spring` directory.

- An XML file that configures the processor component of the application; this file is called `config.xml` and is located in the `META-INF/wlevs` directory

The `config.xml` file configures the `processor1` Oracle CQL processor, in particular the Oracle CQL rules that verify whether the price of a security has fluctuated more than two percent and whether a trend has occurred in its price.

- A Java file that implements the `SignalgenOutputBean` component of the application, a POJO that contains the business logic. This POJO is an `HttpServlet` and an `EventSink`. Its `onEvent` method consumes `PercentTick` and `TrendTick` event instances, computes latency, and displays dashboard information.

In the example, the file is called `SignalgenOutputBean.java` and is located in the `src/oracle/cep/example/signalgen` directory.

For general information about programming event sinks, see [Section , "Handling Events with Sources and Sinks"](#).

- A `MANIFEST.MF` file that describes the contents of the OSGi bundle that will be deployed to Oracle Event Processing.

In the example, the `MANIFEST.MF` file is located in the `META-INF` directory

For more information about creating this file, as well as a description of creating the OSGi bundle that you deploy to Oracle Event Processing, see [Section , "Overview of Application Assembly and Deployment"](#).

- A `dashboard.html` file in the main example directory; this HTML file is the example dashboard that displays events and latencies of the running signal generation application. The HTML file uses Dojo JavaScript libraries from <http://dojotoolkit.org/>, located in the `dojo` directory.



For additional information about the Oracle Event Processing APIs referenced in `ForeignExchangeBuilderFactory`, see *Oracle Fusion Middleware Java API Reference for Oracle Event Processing*.

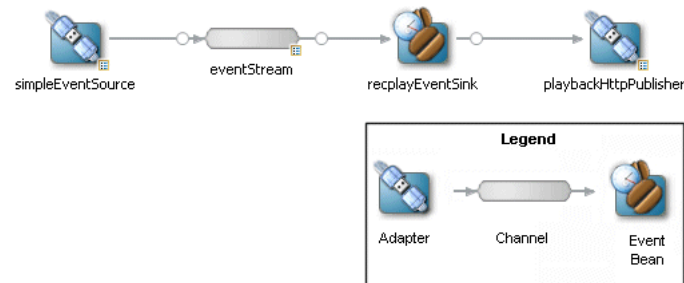
The signal generation example uses a `build.xml` Ant file to compile, assemble, and deploy the OSGi bundle; see [Section , "Building and Deploying the Signal Generation Example from the Source Directory"](#) for a description of this `build.xml` file if you also use Ant in your development environment.

## Event Record and Playback Example

The record and playback example shows how to configure a component to record events to an event store and then configure another component in the network to playback events from the store. The example uses the Oracle Event Processing-provided default Berkeley database to store the events. The example also shows how to configure a publishing HTTP pub-sub adapter as a node in the event processing network.

[Figure 2-78](#) shows the event record and playback example Event Processing Network (EPN). The EPN contains the components that make up the application and how they fit together.

**Figure 2-78 The Event Record and Playback Example Event Processing Network**



The application contains four components in its event processing network:

- `simpleEventSource`: an adapter that generates simple events for purposes of the example. This component has been configured to record events, as shown in the graphic.

The configuration source for this adapter is:

```

<adapter>
  <name>simpleEventSource</name>
  <record-parameters>
    ...
  </record-parameters>
</adapter>
  
```

- `eventStream`: a channel that connects the `simpleEventSource` adapter and `replayEventSink` event bean. This component has been configured to playback events.

The configuration source for this channel is:

```

<channel>
  <name>eventStream</name>
  <playback-parameters>
    ...
  
```

```

    </playback-parameters>
    ...
</channel>

```

- `recplayEventSink`: an event bean that acts as a sink for the events generated by the adapter.
- `playbackHttpPublisher`: a publishing HTTP pub-sub adapter that listens to the `recplayEventSink` event bean and publishes to a channel called `/playbackchannel` of the Oracle Event Processing HTTP Pub-Sub server.

## Running the Event Record/Playback Example

The `recplay_domain` domain contains a single application: the record and playback sample application. To run this application, you first start an instance of Oracle Event Processing in the domain, as described in the following procedure.

The procedure then shows you how to use Oracle Event Processing Visualizer to start the recording and playback of events at the `simpleEventSource` and `eventStream` components, respectively. Finally, the procedure shows you how to use Oracle Event Processing Visualizer to view the stream of events being published to a channel by the `playbackHttpPublisher` adapter.

### To run the event record/playback example:

1. Open a command window and change to the default server directory of the `recplay_domain` domain directory, located in `MIDDLEWARE_HOME\ocep_11.1\samples\domains\recplay_domain\defaultserver`, where `MIDDLEWARE_HOME` refers to the Middleware directory you specified when you installed Oracle Event Processing, such as `d:\Oracle\Middleware`.

```
prompt> cd d:\Oracle\Middleware\ocep_11.1\samples\domains\recplay_
domain\defaultserver
```

2. Set your development environment, as described in [Section , "Setting Your Development Environment."](#)
3. Start Oracle Event Processing by executing the appropriate script with the correct command line arguments:

#### a. On Windows:

- \* If you are using the JRockit JDK included in Oracle JRockit Real Time, enable the deterministic garbage collector by passing the `-dgc` parameter to the command:

```
prompt> startwlevs.cmd -dgc
```

- \* If you are not using the JRockit JDK included in Oracle JRockit Real Time:

```
prompt> startwlevs.cmd
```

#### b. On UNIX:

- \* If you are using the JRockit JDK included in Oracle JRockit Real Time, enable the deterministic garbage collector by passing the `-dgc` parameter to the command:

```
prompt> startwlevs.sh -dgc
```

- \* If you are not using the JRockit JDK included in Oracle JRockit Real Time:

```
prompt> startwlevs.sh
```

After server status messages scroll by, you should see the following message printed to the output:

```
SimpleEvent created at: 14:33:40.441
```

This message indicates that the Oracle Event Processing server started correctly and that the `simpleEventSource` component is creating events.

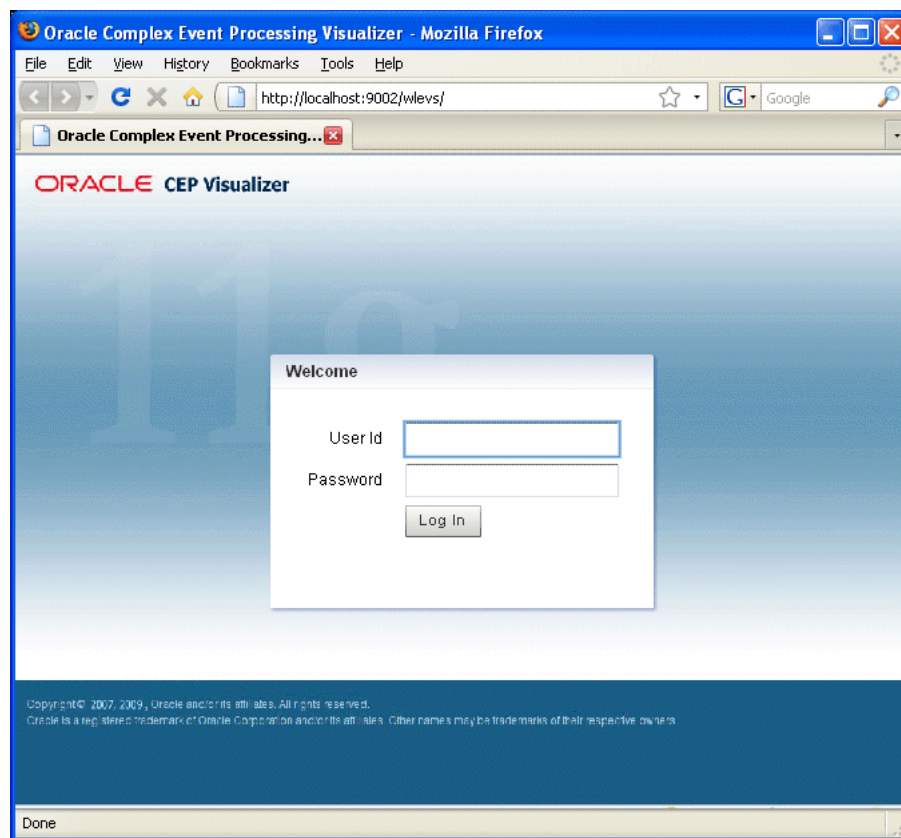
4. Invoke the following URL in your browser:

```
http://host:port/wlevs
```

where *host* refers to the name of the computer on which Oracle Event Processing is running and *port* refers to the Jetty NetIO port configured for the server (default value 9002).

The Logon screen appears as [Figure 2–79](#) shows.

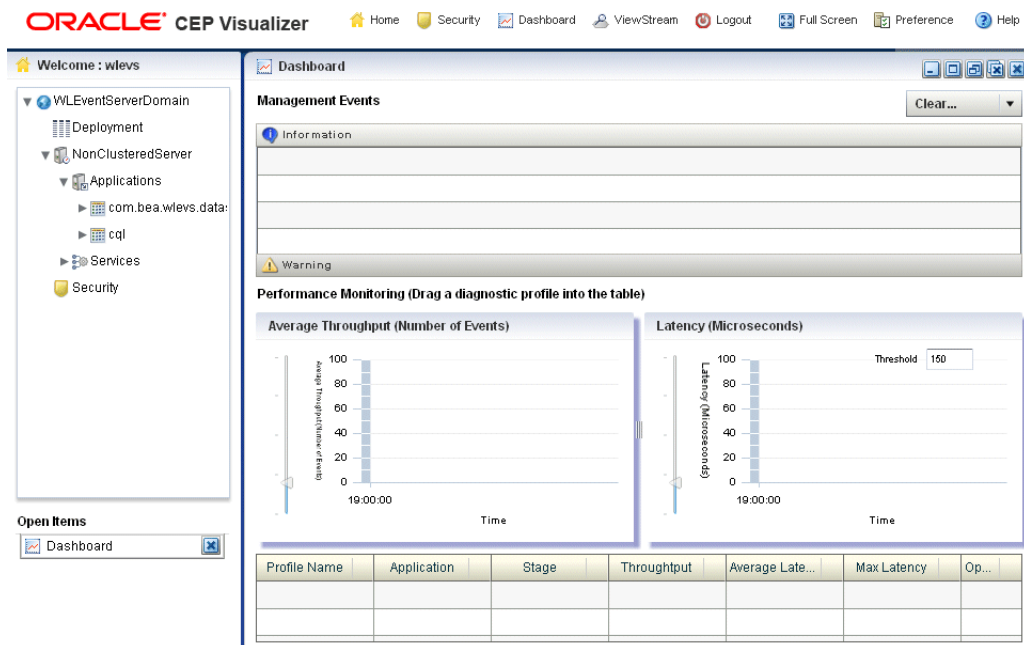
**Figure 2–79 Oracle Event Processing Visualizer Logon Screen**



5. In the Logon screen, enter the **User Id** `wlevs` and **Password** `wlevs`, and click **Log In**.

The Oracle Event Processing Visualizer dashboard appears as [Figure 2–80](#) shows.

**Figure 2–80 Oracle Event Processing Visualizer Dashboard**



For more information about the Oracle Event Processing Visualizer user interface, see "Understanding the Oracle Event Processing Visualizer User Interface" in the *Oracle Fusion Middleware Visualizer User's Guide for Oracle Event Processing*.

6. In the left pane, select **WLEventServerDomain > NonClusteredServer > Applications > replay > Stages > simpleEventSource**.
7. In the right pane, select the **Record** tab as shown in [Figure 2–81](#).

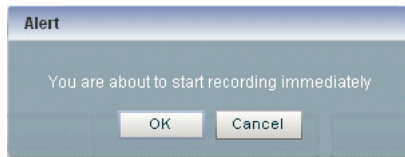
**Figure 2–81 Event Record Tab**

The **DataSet Name** field contains the value of the `record-parameters` child element `dataset-name` element from the `simpleEventSource` adapter application configuration file `MIDDLEWARE_HOME\ocp_11.1\samples\domains\replay_domain\defaultserver\applications\replay\config.xml` as [Example 2–1](#) shows.

**Example 2–1 `replay` Application Configuration File `config.xml`: adapter Element**

```
<adapter>
  <name>simpleEventSource</name>
  <record-parameters>
    <dataset-name>replay_sample</dataset-name>
    <event-type-list>
      <event-type>SimpleEvent</event-type>
    </event-type-list>
    <batch-size>1</batch-size>
    <batch-time-out>10</batch-time-out>
  </record-parameters>
</adapter>
```

8. At the bottom of the Record tab, click **Start**.  
An Alert dialog appears as shown in [Figure 2–82](#).

**Figure 2–82 Start Recording Alert Dialog**

9. Click **OK**.

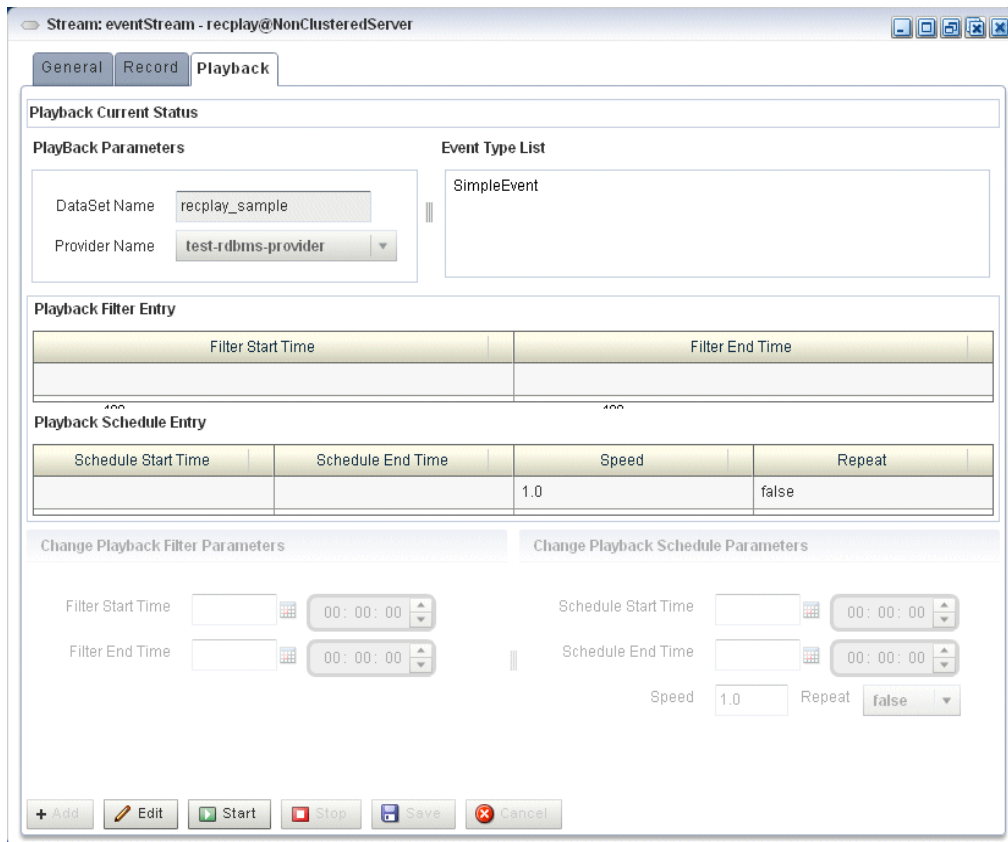
The Current Status field reads **Recording....**

As soon as you click **OK**, events start to flow out of the `simpleEventSource` component and are stored in the configured database.

You can further configure when events are recorded using the **Start Recording** and **Stop Recording** fields.

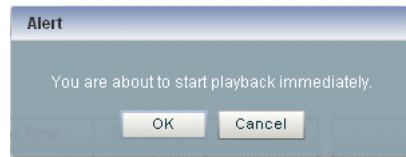
10. In the left pane, select **eventStream**.

11. In the right pane, select the **Playback** tab as shown in [Figure 2–83](#).

**Figure 2–83 Event Playback Tab**

12. At the bottom of the tab, click **Start**.

An Alert dialog appears as shown in [Figure 2–84](#).

**Figure 2–84 Start Playback Alert Dialog****13. Click OK.**

The Current Status field reads **Playing...**

As soon as you click **OK**, events that had been recorded by the `simpleEventSource` component are now played back to the `simpleStream` component.

You should see the following messages being printed to the command window from which you started Oracle Event Processing server to indicate that both original events and playback events are streaming through the EPN:

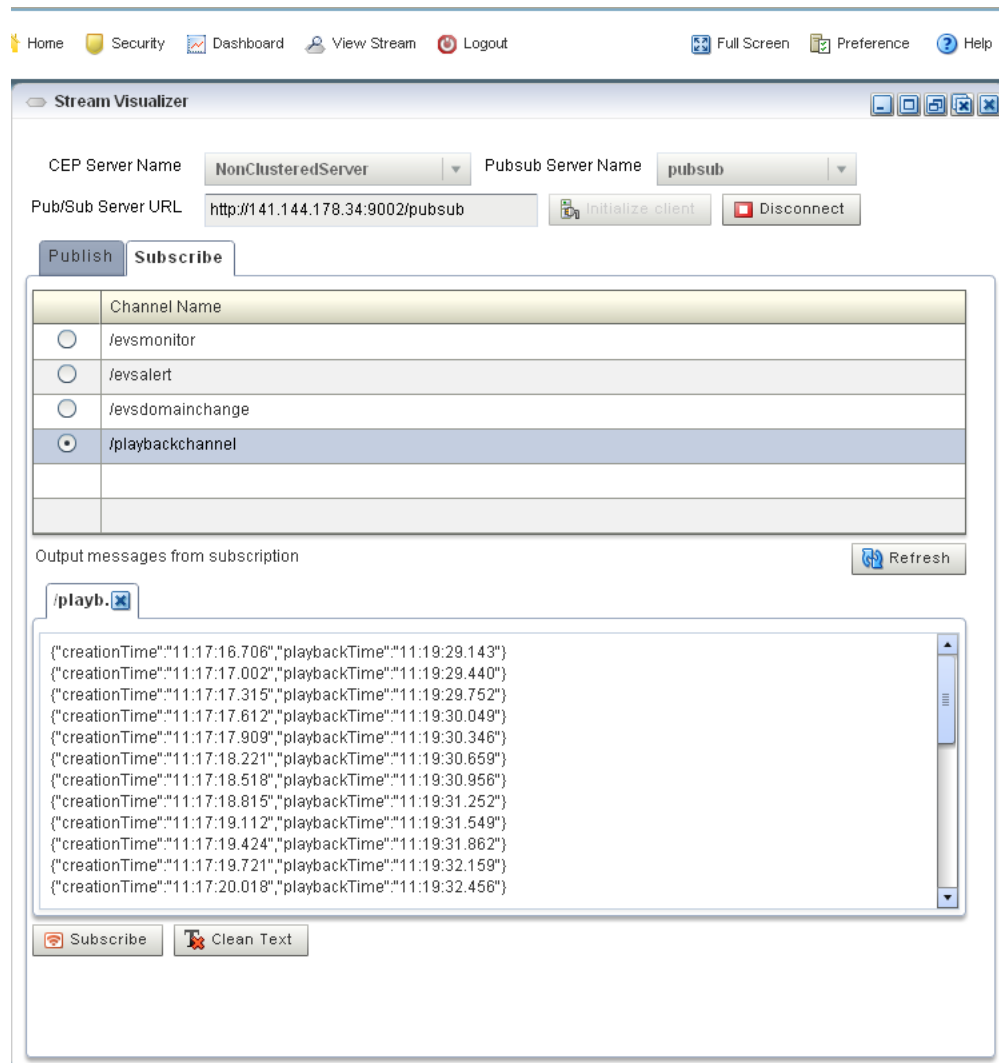
```
SimpleEvent created at: 14:33:11.501
Played back: Original time=14:15:23.141 Playback time=14:33:11.657
```

You can further configure the playback parameters, such as the recorded time period for which you want playback events and the speed that they are played back, by updating the appropriate field and clicking **Change Parameters**. You must restart the playback after changing any playback parameters.

**14. To view the events that the `playbackHttpPublisher` adapter is publishing to a channel, follow these steps:**

- a.** In the top pane, select **Viewstream**.

The Viewstream window appears as shown in [Figure 2–85](#).

**Figure 2–85 Stream Visualizer**

- b. In the right pane, click **Initialize Client**.
- c. In the Subscribe Channel text box, enter `/playbackchannel`.
- d. Click **Subscribe**.

The **Received Messages** text box displays the played back event details. The played back events show the time at which the event was created and the time at which it was played back.

## Building and Deploying the Event Record/Playback Example from the Source Directory

The record and playback sample source directory contains the Java source, along with other required resources, such as configuration XML file and EPN assembly file that make up the application. The `build.xml` Ant file contains targets to build and deploy the application to the `signalgeneration_domain` domain, as described in [Section , "Description of the Ant Targets to Build the Record and Playback Example."](#)



### To build and deploy the event record/playback example from the source directory:

1. If the record/playback Oracle Event Processing instance is not already running, follow the procedure in [Section , "Running the Event Record/Playback Example"](#) to start the server. You must have a running server to successfully deploy the rebuilt application.

2. Open a new command window and change to the record/playback source directory, located in `MIDDLEWARE_HOME\ocep_11.1\samples\source\applications\recplay`, where `MIDDLEWARE_HOME` refers to the Middleware home directory you specified when you installed Oracle Event Processing, such as `d:\Oracle\Middleware`.

```
prompt> cd d:\Oracle\Middleware\ocep_11.1\samples\source\applications\recplay
```

3. Set your development environment, as described in [Section , "Setting Your Development Environment."](#)

4. Execute the `all` Ant target to compile and create the application JAR file:

```
prompt> ant all
```

5. Execute the `deploy` Ant target to deploy the application JAR file to the `MIDDLEWARE_HOME\ocep_11.1\samples\domains\recplay_domain\defaultserver\applications\recplay` directory:

```
prompt> ant -Dusername=wlevs -Dpassword=wlevs -Daction=update deploy
```

---

**Caution:** This target overwrites the existing event record/playback application JAR file in the domain directory.

---

After an application redeploy message, you should see the following message printed to the output about every second:

```
SimpleEvent created at: 14:33:40.441
```

This message indicates that the record and playback example has been redeployed and is running correctly.

6. Follow the instructions in [Section , "Running the Event Record/Playback Example,"](#) starting at step 4, to invoke Oracle Event Processing Visualizer and start recording and playing back events.

## Description of the Ant Targets to Build the Record and Playback Example

The `build.xml` file, located in the top-level directory of the record/playback source, contains the following targets to build and deploy the application:

- `clean`—This target removes the `dist` and output working directories under the current directory.
- `all`—This target cleans, compiles, and jars up the application into a file called `com.bea.wlevs.example.recplay_11.1.1.4_0.jar`, and places the generated JAR file into a `dist` directory below the current directory.
- `deploy`—This target deploys the JAR file to Oracle Event Processing using the Deployer utility.

For more information, see "Deployer Command-Line Reference" in the *Oracle Fusion Middleware Administrator's Guide for Oracle Event Processing*.

## Implementation of the Record and Playback Example

The implementation of the signal generation example generally follows "Creating Oracle Event Processing Applications: Typical Steps" in the *Oracle Fusion Middleware Developer's Guide for Oracle Event Processing for Eclipse*.

Refer to that section for a task-oriented procedure that describes the typical development process.

All the files of the example are located relative to the `MIDDLEWARE_HOME\ocp_11.1\samples\source\applications\recplay` directory, where `MIDDLEWARE_HOME` refers to the Middleware home directory you specified when you installed Oracle Event Processing, such as `c:\Oracle\Middleware`. Oracle recommends that you use this example directory setup in your own environment, although it is obviously not required.

The files used by the record and playback example include:

- An EPN assembly file that describes each component in the application and how all the components are connected together as shown in [Figure 2-78](#).  
  
In the example, the file is called `com.bea.wlevs.example.recplay-context.xml` and is located in the `META-INF/spring` directory.
- Java source file for the `simpleEventSource` adapter.  
  
In the example, the file is called `SimpleEventSource.java` and is located in the `src/com/bea/wlevs/adapter/example/recplay` directory.  
  
For a detailed description of how to program the adapter Java files in general, see [Section , "Overview of Custom Adapters"](#).
- Java source file that describes the `PlayedBackEvent` and `SimpleEvent` event types. The `SimpleEvent` event type is the one originally generated by the adapter, but the `PlayedBackEvent` event type is used for the events that are played back after having been recorded. The `PlayedBackEvents` look almost exactly the same as `SimpleEvent` except they have an extra field, the time the event was recorded.  
  
In the example, the two events are called `SimpleEvent.java` and `PlayedBackEvent.java` and are located in the `src/com/bea/wlevs/event/example/recplay` directory.  
  
For a detailed description of this file, as well as general information about programming event types, see [Section , "Overview of Oracle Event Processing Event Types"](#).
- A Java file that implements the `recplayEventSink` event bean of the application, which is an event sink that receives both realtime events from the `simpleEventSource` adapter as well as playback events.  
  
In the example, the file is called `RecplayEventSink.java` and is located in the `src/com/bea/wlevs/example/recplay` directory.  
  
For more information about event sources and sinks, see [Section , "Handling Events with Sources and Sinks"](#).
- An XML file that configures the `simpleEventSource` adapter and `eventStream` channel components. The adapter includes a `<record-parameters>` element that specifies that the component will record events to the event store; similarly, the

channel includes a `<playback-parameters>` element that specifies that it receives playback events.

In the example, the file is called `config.xml` and is located in the `META-INF/wlevs` directory.

- A `MANIFEST.MF` file that describes the contents of the OSGi bundle that will be deployed to Oracle Event Processing.

In the example, the `MANIFEST.MF` file is located in the `META-INF` directory

For more information about creating this file, as well as a description of creating the OSGi bundle that you deploy to Oracle Event Processing, see [Section , "Overview of Application Assembly and Deployment"](#).

The record/playback example uses a `build.xml` Ant file to compile, assemble, and deploy the OSGi bundle; see [Section , "Building and Deploying the Event Record/Playback Example from the Source Directory"](#) for a description of this `build.xml` file if you also use Ant in your development environment.



---

# Getting Started with Developing Oracle Event Processing Applications

This chapter provides suggestions for getting started in building Oracle Event Processing applications, including suggested start-to-finish steps, setting up a development environment, and tools for development and testing.

This chapter includes the following sections:

- [Creating an Oracle Event Processing Application](#)
- [Setting Your Development Environment](#)
- [Using an IDE to Develop Applications](#)
- [Testing Applications](#)

## Creating an Oracle Event Processing Application

The following procedure shows the *suggested* start-to-finish steps to create an Oracle Event Processing 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 Event Processing developers, see [Chapter 4, "Overview of the Oracle Event Processing IDE for Eclipse"](#)

### To create an Oracle Event Processing application:

1. Set up your environment as described in [Section , "Setting Your Development Environment."](#)
2. Create an Oracle Event Processing project using the Oracle Event Processing IDE for Eclipse.

For more information, see [Chapter 5, "Oracle Event Processing IDE for Eclipse Projects"](#).

3. Design your event processing network (EPN).

Using the Oracle Event Processing IDE for Eclipse and the EPN editor, add 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 (the EPN assembly file) and the XML file that specifies component configuration (the component configuration file). You

will likely be constantly updating these XML files as you implement your application, but Oracle recommends you start with this step so you have a high-level view of your application.

For more information, see:

- [Chapter 7, "Oracle Event Processing IDE for Eclipse and the Event Processing Network"](#)
  - [Section , "Creating EPN Assembly Files."](#)
  - [Section , "Creating Component Configuration Files."](#)
4. Determine the event types that your application is going to use, and, if creating your own JavaBean, program the Java file.  
See [Chapter 9, "Defining and Using 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 Event Processing. For details, see:
    - [Section , "Oracle Event Processing APIs"](#)
    - [Section , "Configuring Oracle Event Processing Resource Access"](#)
    - [Chapter 11, "Integrating the Java Message Service"](#)
    - [Chapter 12, "Integrating an HTTP Publish-Subscribe Server"](#)
    - [Chapter 15, "Integrating an External Component Using a Custom Adapter"](#)
    - [Chapter 16, "Handling Events with Java"](#)
  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 17, "Querying an Event Stream with Oracle CQL"](#)
    - [Chapter 19, "Querying an Event Stream with Oracle EPL"](#)
  7. Design the rules that the processors are going to use to select events from their upstream channels and output events to their downstream channels.  
See:
    - *Oracle Fusion Middleware CQL Language Reference for Oracle Event Processing*
    - *Oracle Fusion Middleware EPL Language Reference for Oracle Event Processing*

---

**Note:** Oracle CQL replaces Event Processing Language (EPL) in Oracle Event Processing 11g Release 1 (11.1.1). Oracle Event Processing supports EPL for backwards compatibility.

---
  8. Optionally configure the channels that stream data between adapters, processors, and the business logic POJO by creating their configuration XML files.  
See [Chapter 10, "Connecting EPN Stages Using Channels."](#)

9. 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 13, "Integrating a Cache."](#)

10. Optionally, use the Oracle Event Processing server log subsystem to write log messages from your application to the Oracle Event Processing 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 Event Processing Visualizer, you can deploy your application, configure the log level for your application, and view the Oracle Event Processing server console.

For more information, see:

- "Configuring Logging and Debugging for Oracle Event Processing" in the *Oracle Fusion Middleware Administrator's Guide for Oracle Event Processing*
- [Section , "Managing Log Message Catalogs"](#)
- "How to Configure Component Logging" in the *Oracle Fusion Middleware Visualizer User's Guide for Oracle Event Processing*
- "How to View Console Output" in the *Oracle Fusion Middleware Visualizer User's Guide for Oracle Event Processing*

## Setting Your Development Environment

You must set your development environment before you can start Oracle Event Processing instances and run the samples. In particular, you must set the `PATH` and `JAVA_HOME` environment variables so that you are using the correct version of the JRockit JDK.

There are two ways in which JRockit might have been installed on your computer:

- As part of the Oracle JRockit Real Time installation. This version of the JRockit JDK includes the deterministic garbage collector.
- As part of the Oracle Event Processing 11g Release 1 (11.1.1) installation. This version of the JRockit JDK *does not* include the deterministic garbage collector, and is provided for testing purposes only.

Although not required, Oracle recommends that you run Oracle Event Processing using the JRockit JDK version included in Oracle JRockit Real Time for best results; however, the following procedures describe how to set your environment for either case.

For more information about JRockit, see [Section , "Increasing the Performance of the Samples"](#).

This section describes:

- [Section , "How to Set Your Development Environment on Windows"](#)
- [Section , "How to Set Your Development Environment on UNIX"](#)

## How to Set Your Development Environment on Windows

This procedure describes how to set your development environment on Windows.

To make it easier to reset your development environment after logging out of a session, you can create a command file, such as `setEnv.cmd`, that contains the `set` commands this section describes.

You can also set the required environment variables permanently on your Windows computer by invoking the **Control Panel > System** window, clicking the **Advanced** tab, and then clicking the **Environment Variables** button. You can set the environment variables for the current user or for the entire system.

### To set your development environment on Windows:

1. Update your `PATH` environment variable to include the `bin` directory of the JRockit JDK. Also, be sure that your `PATH` environment variable includes the `bin` directory of your Ant installation:

- a. If using the JRockit JDK installed with Oracle JRockit Real Time:

If you installed Oracle JRockit Real Time in the `d:\jrockit` directory and Ant is installed in the `d:\ant` directory, set your `PATH` environment variable as shown:

```
prompt> set PATH=d:\jrockit\[JRRT_HOME]\bin;d:\ant\bin;%PATH%
```

where `JRRT_HOME` is the JRockit Real Time directory.

- b. If using the JRockit JDK installed with Oracle Event Processing:

If you installed Oracle Event Processing in the `d:\Oracle\Middleware` directory and Ant is installed in the `d:\ant` directory, set your `PATH` environment variable as shown:

```
prompt> set PATH=d:\Oracle\Middleware\jrockit_160_20\bin;d:\ant\bin;%PATH%
```

2. Ensure that the `JAVA_HOME` variable in the `setDomainEnv.cmd` script points to the correct JRockit JDK. If it does not, edit the script.

The `setDomainEnv.cmd` script is located in the `defaultserver` subdirectory of the main domain directory; the `defaultserver` subdirectory contains the files for the standalone server of each domain. For example, the `HelloWorld` domain is located in `MIDDLEWARE_HOME\ocep_11.1\samples\domains\helloworld_domain`, where `MIDDLEWARE_HOME` refers to the Middleware home directory you specified when you installed Oracle Event Processing, such as `d:\Oracle\Middleware`.

- a. If using the JRockit JDK installed with Oracle JRockit Real Time:

The `set` command should be as follows:

```
set JAVA_HOME=d:\jrockit\[JRRT_HOME]
```

where `JRRT_HOME` is the JRockit Real Time directory.

- b. If using the JRockit JDK installed with Oracle Event Processing:

The `set` command should be as follows:

```
set JAVA_HOME=d:\Oracle\Middleware\jrockit_160_20
```

3. Set the `JAVA_HOME` variable in your own development environment to point to the JRockit JDK.

- a. If using the JRockit JDK installed with Oracle JRockit Real Time:



The set command should be as follows:

```
prompt> set JAVA_HOME=d:\jrockit\[JRRT_HOME]
```

where JRRT\_HOME is the JRockit Real Time directory.

- b. If using the JRockit JDK installed with Oracle Event Processing:

The set command should be as follows:

```
prompt> set JAVA_HOME=d:\Oracle\Middleware\jrockit_160_20
```

## How to Set Your Development Environment on UNIX

This procedure describes how to set your development environment on UNIX.

To make it easier to reset your development environment after logging out of a session, you can create a command file, such as `setEnv.sh`, that contains the set commands this section describes.

### To set your development environment on UNIX:

1. Update your `PATH` environment variable to include the `bin` directory of the JRockit JDK. Also, be sure that your `PATH` environment variable includes the `bin` directory of your Ant installation.

- a. If using the JRockit JDK installed with Oracle JRockit Real Time:

If you installed Oracle JRockit Real Time in the `/jrockit` directory and Ant is installed in the `/ant` directory, set your `PATH` environment variable as follows:

```
prompt> PATH=/jrockit/j\[JRRT_HOME]/bin:/ant/bin:$PATH
```

where JRRT\_HOME is the JRockit Real Time directory.

- b. If using the JRockit JDK installed with Oracle Event Processing:

If you installed Oracle Event Processing in the `/Oracle/Middleware` directory and Ant is installed in the `/ant` directory, set your `PATH` environment variable as shown:

```
prompt> PATH=/Oracle/Middleware/jrockit_160_20/bin:/ant/bin:$PATH
```

2. Ensure that the `JAVA_HOME` variable in the `setDomainEnv.sh` script points to the correct JRockit JDK. If it does not, edit the script.

The `setDomainEnv.sh` script is located in the `defaultserver` subdirectory of the main domain directory; the `defaultserver` subdirectory contains the files for the standalone server of each domain. For example, the `HelloWorld` domain is located in `MIDDLEWARE_HOME/ocp_11.1/samples/domains/helloworld_domain`, where `MIDDLEWARE_HOME` refers to the Middleware home directory you specified when you installed Oracle Event Processing, such as `/Oracle/Middleware`.

- a. If using the JRockit JDK installed with Oracle JRockit Real Time:

The `JAVA_HOME` variable should be set as follows:

```
JAVA_HOME=/jrockit/[JRRT_HOME]
```

where JRRT\_HOME is the JRockit Real Time directory.

- b. If using the JRockit JDK installed with Oracle Event Processing:

The `JAVA_HOME` variable should be set as follows:

```
JAVA_HOME=/Oracle/Middleware/jrockit_160_20
```

3. Set the `JAVA_HOME` variable in your development environment to point to the JRockit JDK.

- a. If using the JRockit JDK installed with Oracle JRockit Real Time:

The `JAVA_HOME` variable should be set as follows:

```
prompt> JAVA_HOME=/jrockit/[JRRT_HOME]
```

where `JRRT_HOME` is the JRockit Real Time directory.

- b. If using the JRockit JDK installed with Oracle Event Processing:

The `JAVA_HOME` variable should be set as follows:

```
prompt> JAVA_HOME=/Oracle/Middleware/jrockit_160_20
```

## Using an IDE to Develop Applications

Oracle Event Processing includes a plugging that enhances the Eclipse IDE with features specifically designed to ease the work of building Oracle Event Processing applications. For more information about the IDE, see [Section 4, "Overview of the Oracle Event Processing IDE for Eclipse"](#).

## Testing Applications

You can test Oracle Event Processing applications you build by using the included `csvadapter` and `load generator`.

The `load generator` is a command-line tool that reads a comma-separated values (CSV) file and feeds the results to the `csvadapter`. The `csvadapter`, in turn, is designed to receive values from the `load generator`, then bind those values to an event type that you specify.

Using these tools is a relatively simple way to try out code in development before you are ready for the application to receive data from the actual event data source.

For more information about testing with these tools, see [Section 21, "Testing Applications With the Load Generator and `csvgen` Adapter"](#).

# Part II

---

## Oracle Event Processing IDE for Eclipse

Part II contains the following chapters:

- [Chapter 4, "Overview of the Oracle Event Processing IDE for Eclipse"](#)
- [Chapter 5, "Oracle Event Processing IDE for Eclipse Projects"](#)
- [Chapter 6, "Oracle Event Processing IDE for Eclipse and Oracle Event Processing Servers"](#)
- [Chapter 7, "Oracle Event Processing IDE for Eclipse and the Event Processing Network"](#)



---

---

# Overview of the Oracle Event Processing IDE for Eclipse

This chapter introduces the Oracle Event Processing IDE features available for Eclipse, providing information on how to install and configure the features.

This chapter includes the following sections:

- [Overview of Oracle Event Processing IDE for Eclipse](#)
- [Installing the Latest Oracle Event Processing IDE for Eclipse](#)
- [Installing the Oracle Event Processing IDE for Eclipse Distributed With Oracle Event Processing](#)
- [Configuring Eclipse](#)

## Overview of Oracle Event Processing IDE for Eclipse

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

This section describes:

- [Section , "Features"](#)
- [Section , "JDK Requirements"](#)
- [Section , "Default Oracle Event Processing Domain ocep\\_domain and Development"](#)

For more information about Oracle Event Processing IDE for Eclipse, see <http://www.oracle.com/technology/products/event-driven-architecture/cep-ide/11/>.

## Features

The key features of the Oracle Event Processing 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 Event Processing applications.
- Integrated server management to seamlessly start, stop, and deploy to Oracle Event Processing 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 Event Processing 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 Event Processing applications.
- Integrated support for the Oracle Event Processing Visualizer so you can use the Oracle Event Processing Visualizer from within the IDE.

## JDK Requirements

In 11g Release 1 (11.1.1), Oracle Event Processing IDE for Eclipse requires JDK 6.0. For more information, see:

- [Section , "Setting Your Development Environment."](#)
- [Section , "Configuring Eclipse"](#)

## Default Oracle Event Processing Domain ocep\_domain and Development

When you choose a **Typical** Oracle Event Processing server install, the installation does not include the default `ocep_domain` domain (with default passwords) and the product samples.

If you want to install the default `ocep_domain` and samples (recommended), choose the **Custom** Oracle Event Processing server install option.

The **Typical** install is appropriate for a production environment while the **Custom** install is appropriate for a development environment.

Oracle recommends that you install the default `ocep_domain` and samples for use with the Oracle Event Processing IDE for Eclipse during development.

If you choose a **Typical** Oracle Event Processing server install, you can use the Configuration Wizard to create an Oracle Event Processing server domain.

For more information, see:

- "Installation Overview" in the *Oracle Fusion Middleware Getting Started Guide for Oracle Event Processing*
- "Creating an Oracle Event Processing Standalone-Server Domain" in the *Oracle Fusion Middleware Administrator's Guide for Oracle Event Processing*
- "Creating an Oracle Event Processing Multi-Server Domain Using Oracle Event Processing Native Clustering" in the *Oracle Fusion Middleware Administrator's Guide for Oracle Event Processing*

## Installing the Latest Oracle Event Processing 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.

### To install the latest Oracle Event Processing IDE for Eclipse:

1. Obtain the required versions of Eclipse (3.7.2) and WTP (2.0). Be sure to install the Eclipse IDE for Java EE developers. We recommend you take the entire Indigo installation available at the following Web sites:

**Windows:**

<http://www.eclipse.org/downloads/download.php?file=/technology/epp/downloads/release/indigo/R/eclipse-jee-indigo-win32.zip>

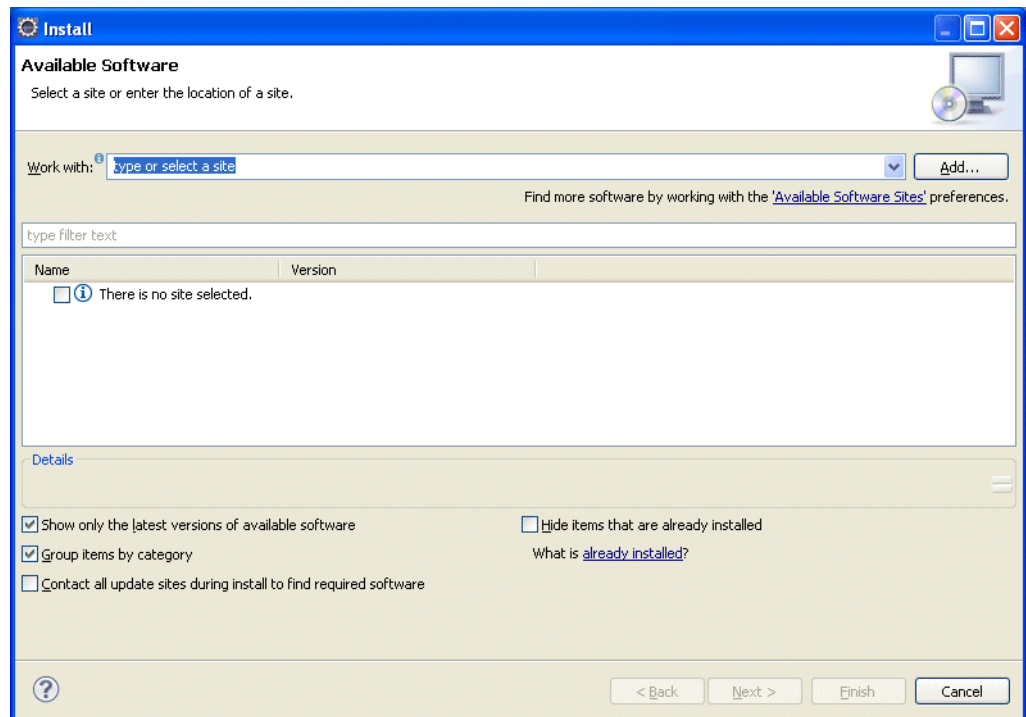
**Linux:**

<http://www.eclipse.org/downloads/download.php?file=/technology/epp/downloads/release/indigo/SR2/eclipse-jee-indigo-SR2-linux-gtk.tar.gz>

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

The Install dialog appears as [Figure 4-1](#) shows.

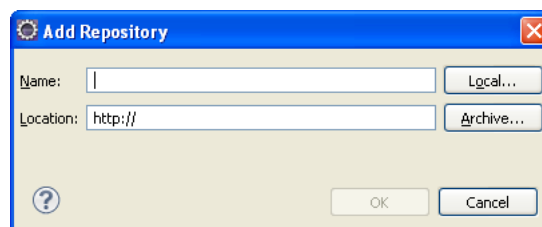
**Figure 4-1 Install Dialog**



3. Click **Add**.

The Add Site dialog appears as [Figure 4-2](#) shows.

**Figure 4-2 Add Site Dialog**

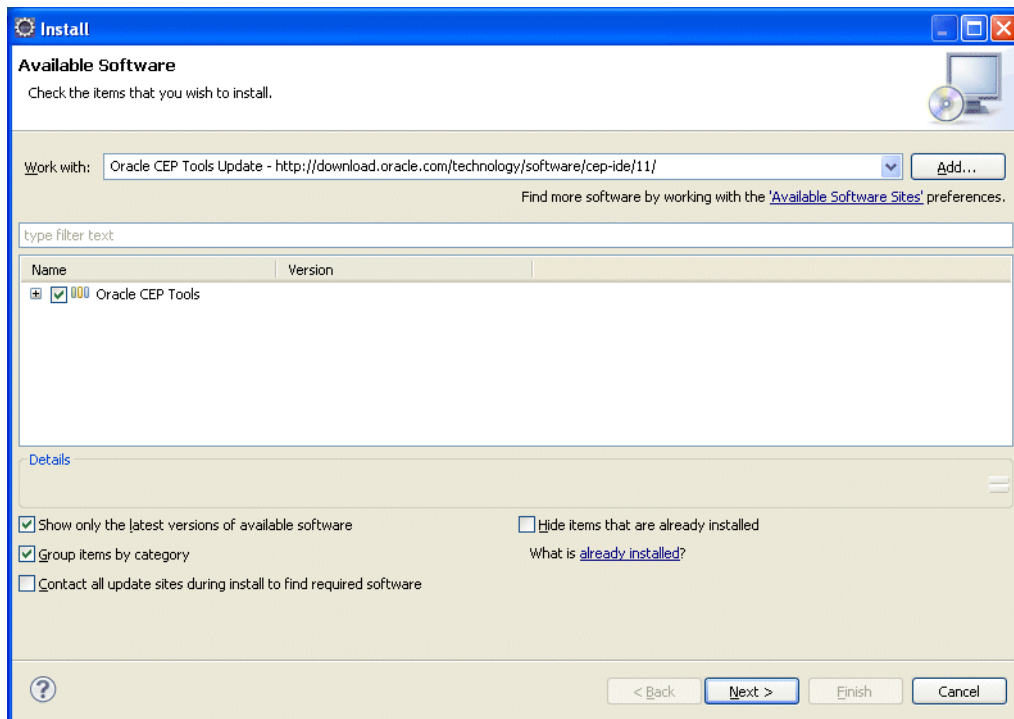


4. Configure this dialog as [Table 4-1](#) describes.

**Table 4–1 New Update Site Dialog Attributes**

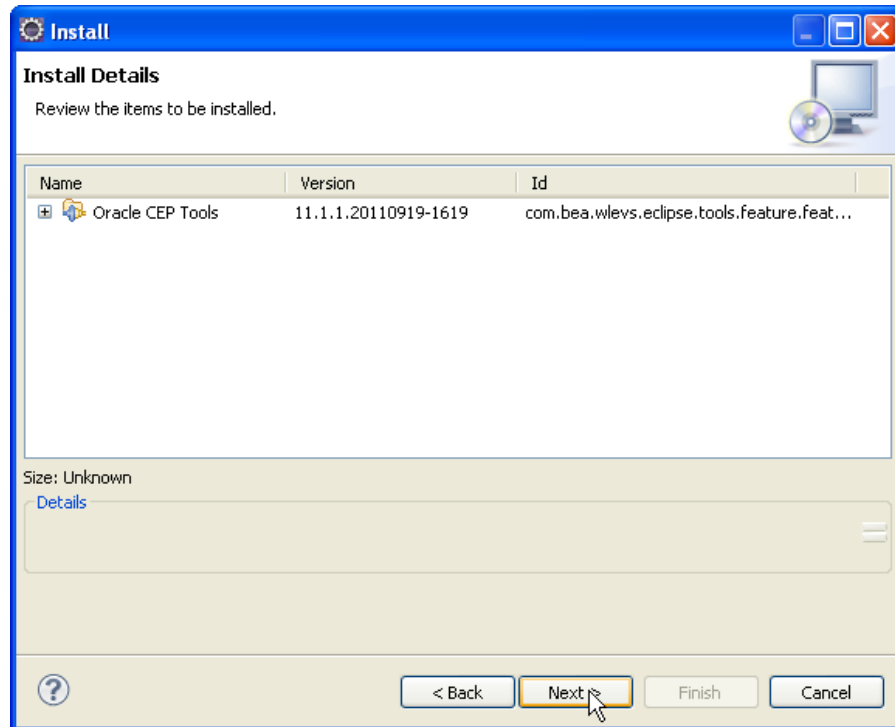
Attribute	Description
Name	The name for this remote update site. For example: Oracle Event Processing Tools Update.
URL	The URL to the remote update site. Valid value: http://download.oracle.com/technology/software/cep-ide/11/

5. Click **OK**.
6. In the Install dialog, from the **Work with** pull down menu, select the Oracle Event Processing Tools Update site you just created.  
  
It may take a few moments for Eclipse to contact the remote update site. During this time, the "There is no site selected" entry reads "Pending".  
  
When Eclipse has made contact with the remote update site, the Oracle Event Processing Tools entry appears in the list of update sites as [Figure 4–3](#) shows.
7. Check the check box next to the **Oracle Event Processing Tools** entry as [Figure 4–3](#) shows.

**Figure 4–3 Install Dialog - Site Selected**

8. Click **Next**.  
  
The Install Details dialog appears as [Figure 4–4](#) shows.



**Figure 4–4 Install Dialog - Install Details****9. Click Next.**

The Review Licenses dialog appears.

**10. Click Finish.**

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

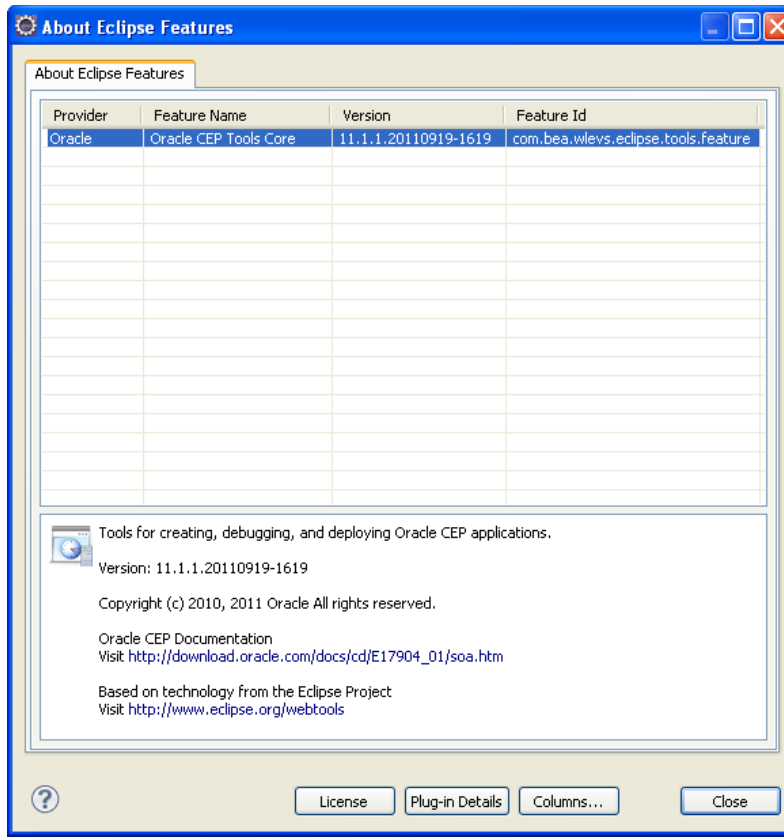
**12. To confirm the installation, select Help > About Eclipse.**

The About Eclipse dialog appears as [Figure 4–5](#) shows.

**Figure 4–5 About Eclipse****13. Click Oracle.**

The About Eclipse Features dialog appears as [Figure 4–6](#) shows.

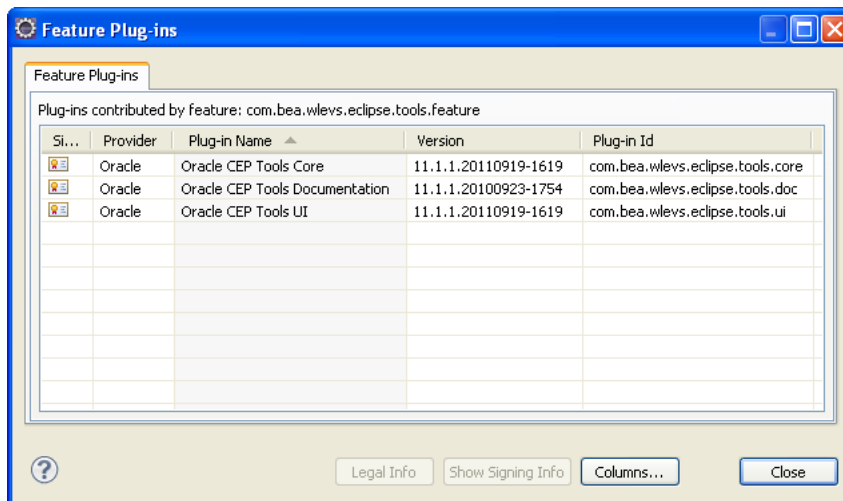
**Figure 4–6 About Eclipse Features Dialog**



**14. Click Plug-In Details.**

The Feature Plug-ins dialog appears as [Figure 4–7](#) shows.

**Figure 4–7 Feature Plug-ins Dialog**



**15. Confirm that the plug-ins that [Table 4–2](#) lists are shown.**

**Table 4–2 Oracle Event Processing IDE for Eclipse Plug-Ins**

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

16. After installing Oracle Event Processing IDE for Eclipse, consider the following topics:
- [Section , "Default Oracle Event Processing Domain ocep\\_domain and Development"](#)
  - [Section , "Configuring Eclipse"](#)

## Installing the Oracle Event Processing IDE for Eclipse Distributed With Oracle Event Processing

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

### To install the Oracle Event Processing IDE for Eclipse distributed with Oracle Event Processing:

1. Obtain the required versions of Eclipse (3.7.2) and WTP (2.0). Be sure to install the Eclipse IDE for Java EE developers. We recommend you take the entire Indigo installation available at the following Web sites:

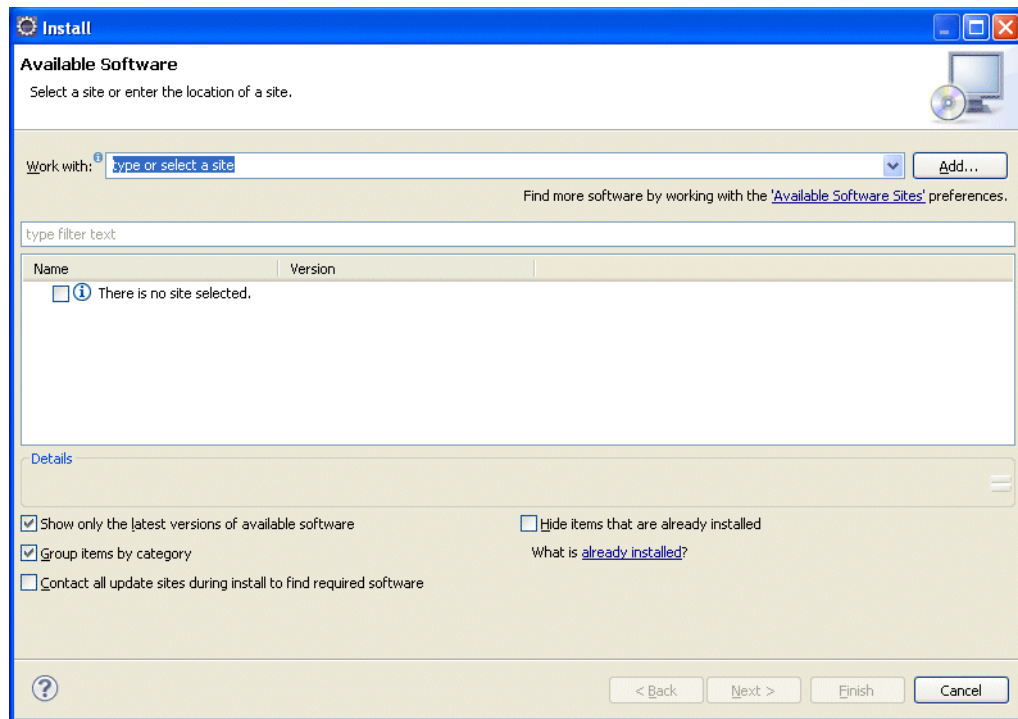
**Windows:**

<http://www.eclipse.org/downloads/download.php?file=/technology/epp/downloads/release/indigo/R/eclipse-jee-indigo-win32.zip>

**Linux:**

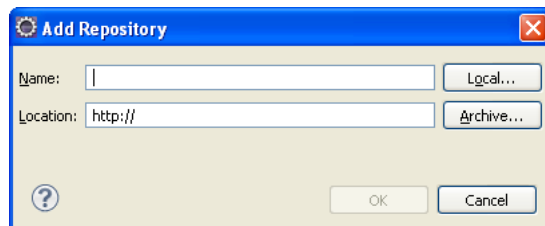
<http://www.eclipse.org/downloads/download.php?file=/technology/epp/downloads/release/indigo/SR2/eclipse-jee-indigo-SR2-linux-gtk.tar.gz>

2. Open your Eclipse IDE and select the menu item **Help > Install New Software**. The Install dialog appears as [Figure 4–1](#) shows.

**Figure 4–8 Install Dialog**

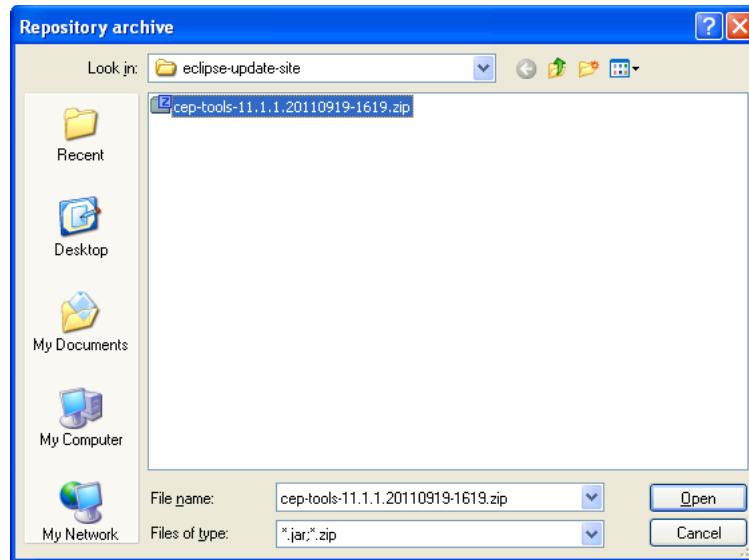
3. Click **Add**.

The Add Site dialog appears as [Figure 4–2](#) shows.

**Figure 4–9 Add Site Dialog**

4. Click **Archive**.

The Select Local Site Archive dialog appears as [Figure 4–10](#) shows.

**Figure 4–10 Select Local Site Archive Dialog**

5. Navigate to the `MIDDLEWARE_HOME/ocep_11.1/eclipse-update-site` directory and select the `cep-tools-11.1.0.DATE-BUILD.zip` file.

Where `MIDDLEWARE_HOME` refers to the directory into which you installed Oracle Event Processing, such as `c:\Oracle\Middleware`, and `DATE` is the build date and `BUILD` is the build number.

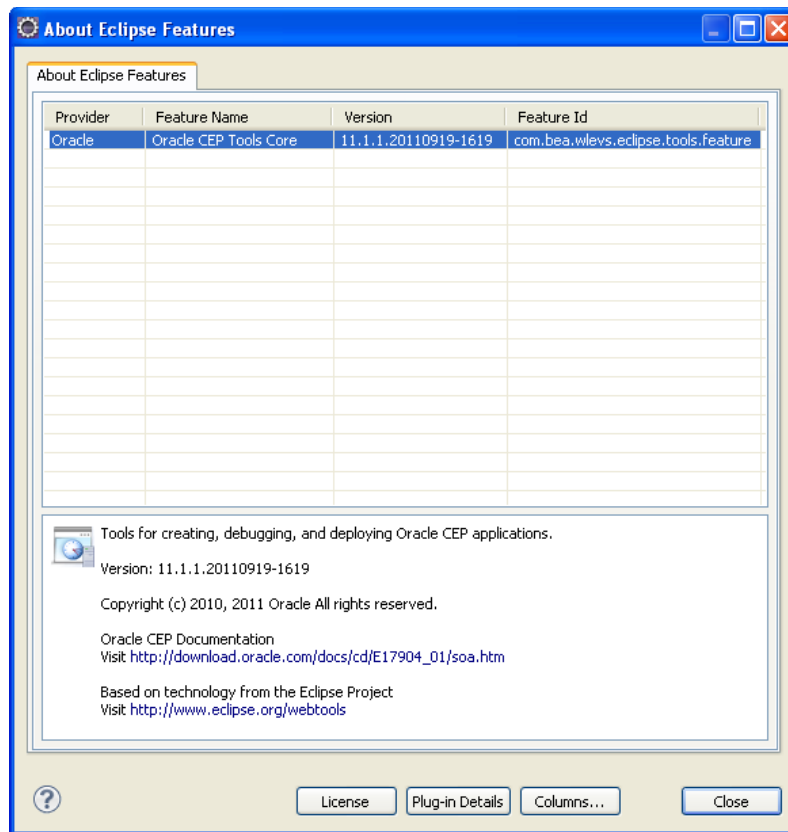
6. Click **Open**.
7. Complete the Update Manager, selecting to install the Oracle Event Processing tools.
8. When prompted restart, Eclipse. If you skip this, unreliable behavior can occur.
9. To confirm the installation, select **Help > About Eclipse**.

The About Eclipse dialog appears as [Figure 4–5](#) shows.

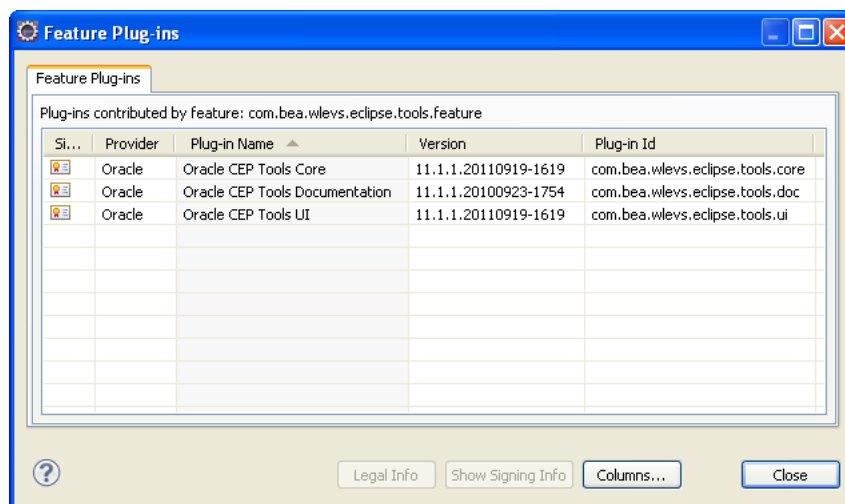
**Figure 4–11 About Eclipse**

10. Click **Oracle**.

The About Eclipse Features dialog appears as [Figure 4–6](#) shows.

**Figure 4–12 About Eclipse Features Dialog****11. Click Plug-In Details.**

The Feature Plug-ins dialog appears as [Figure 4–7](#) shows.

**Figure 4–13 Feature Plug-ins Dialog****12. Confirm that the plug-ins that [Table 4–2](#) lists are shown.**

**Table 4–3 Oracle Event Processing IDE for Eclipse Plug-Ins**

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

13. After installing Oracle Event Processing IDE for Eclipse, consider the following topics:
- [Section , "Default Oracle Event Processing Domain ocep\\_domain and Development"](#)
  - [Section , "Configuring Eclipse"](#)

## Configuring Eclipse

This section describes how to configure Eclipse to work with Oracle Event Processing.

### To configure Eclipse:

1. Exit out of Eclipse if it is running.
2. Install a Java 6 JRE on your computer.

For example, you might have installed the JRE included with a Java Development Kit to the following location:

```
C:\Java\jre6
```

3. Using the editor of your choice, open your `eclipse.ini` file located in your Eclipse install directory, for example, `C:\eclipse\` as [Example 4–1](#) shows.

#### **Example 4–1 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](http://wiki.eclipse.org/FAQ_How_do_I_run_Eclipse).

---

4. Add the following lines to the `eclipse.ini` file as [Example 4–2](#) shows.

#### **Example 4–2 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 4–3](#) shows.

**Example 4–3 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:\Java\jre6\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 4–3](#) shows.

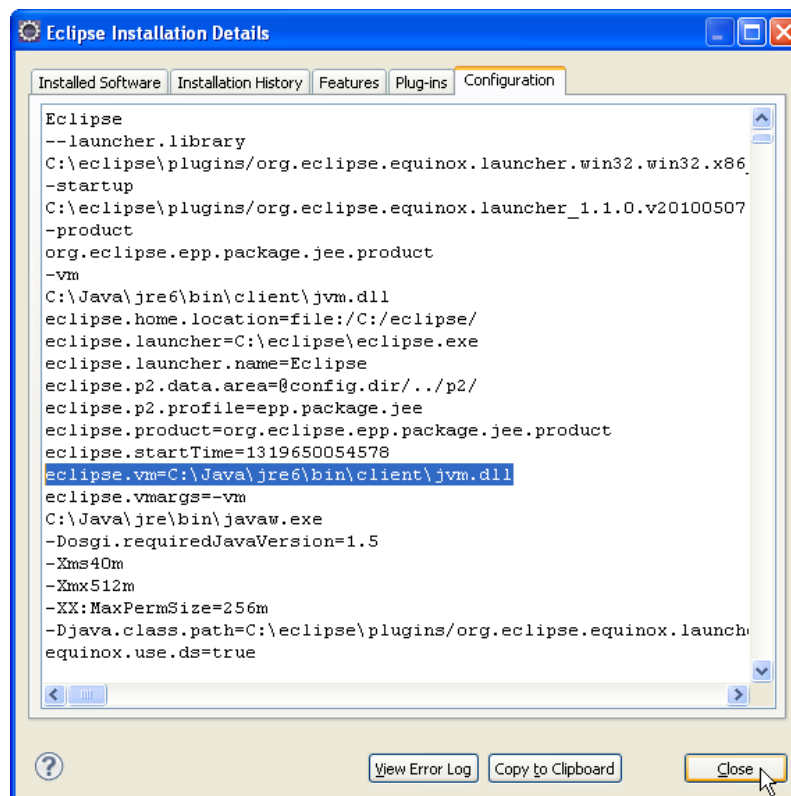
---

---

6. Save and close the `eclipse.ini` file.
7. Start Eclipse.
8. Select **Help > About Eclipse** and click **Installation Details**.
9. Click the **Configuration** tab.

The Configuration Details tab appears as shown in [Figure 4–14](#).



**Figure 4–14 Configuration Details for Java 6**

10. Confirm that the `eclipse.vm` property points to the Java 6.0 JRE you configured in the `eclipse.ini` file.



---

# Oracle Event Processing IDE for Eclipse Projects

This chapter describes how to use the Oracle Event Processing IDE for Eclipse to create projects with which to develop event processing networks (EPNs), including EPN assembly files and component configuration files.

This chapter includes the following sections:

- [Oracle Event Processing Project Overview](#)
- [Creating Oracle Event Processing Projects](#)
- [Creating EPN Assembly Files](#)
- [Creating Component Configuration Files](#)
- [Exporting Oracle Event Processing Projects](#)
- [Upgrading Projects](#)
- [Managing Libraries and Other Non-Class Files in Oracle Event Processing Projects](#)
- [Configuring Oracle Event Processing IDE for Eclipse Preferences](#)

## Oracle Event Processing Project Overview

An Oracle Event Processing application includes the following artifacts:

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

[Figure 5-1](#) shows the Explorer after creating a project.

**Figure 5–1 Oracle Event Processing Project Structure**

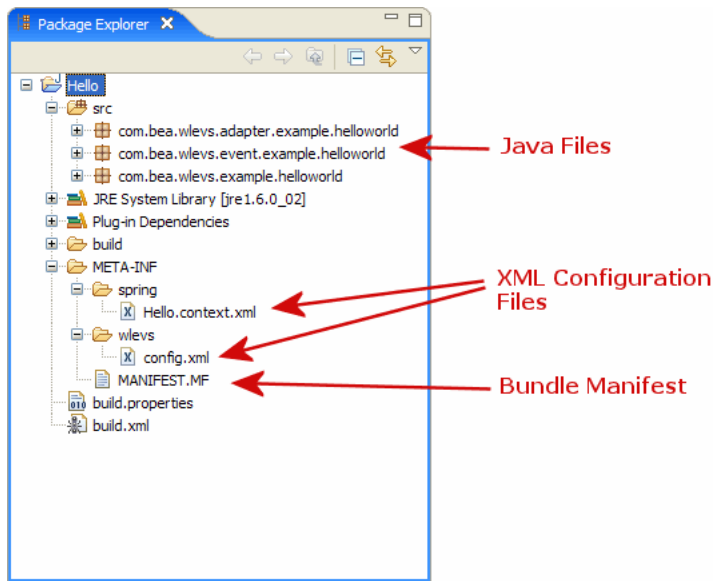


Table 5–1 summarizes the important files in an Oracle Event Processing project including their use and location.

**Table 5–1 Oracle Event Processing Project Artifacts**

File Type	Location	Description
Java source files	Any Java source folder. Default: <code>src</code> .	Events, adapters, and listeners are implemented in an Oracle Event Processing application with Java files. All Java files must be in a source folder in order to be compiled.  For more information, see <a href="#">Chapter 1, "Overview of Creating Oracle Event Processing Applications"</a> .
EPN assembly file	<code>META-INF/spring</code>	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 <a href="#">Chapter , "Overview of EPN Assembly Files"</a> .
Processor configuration file	<code>META-INF/wlevs</code>	The processor configuration file is where the Oracle Event Processing processor 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: <ul style="list-style-type: none"> <li>▪ <a href="#">Chapter 17, "Querying an Event Stream with Oracle CQL"</a></li> <li>▪ <a href="#">Chapter 19, "Querying an Event Stream with Oracle EPL"</a></li> </ul>
MANIFEST.MF file	<code>META-INF</code>	The manifest file contains metadata about your application including its name, version, and dependencies, among others.  For more information, see <a href="#">Chapter 23, "Assembling and Deploying Oracle Event Processing Applications"</a> .

## Creating Oracle Event Processing Projects

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

Projects correspond 1-to-1 with Oracle Event Processing 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 Event Processing application.

## How to Create an Oracle Event Processing Project

By default new projects are set to use Java 6.0. This section describes how to create an Oracle Event Processing project using Java 6. For information on configuring an Oracle Event Processing project to use Java 6, see [Section , "Configuring Eclipse"](#).

---

**Note:** By default, the Overview view of the Eclipse manifest editor provides a link to launch the OSGi framework for testing the plugin. When you've created an Oracle Event Processing project using an application template, launching the framework is not supported.

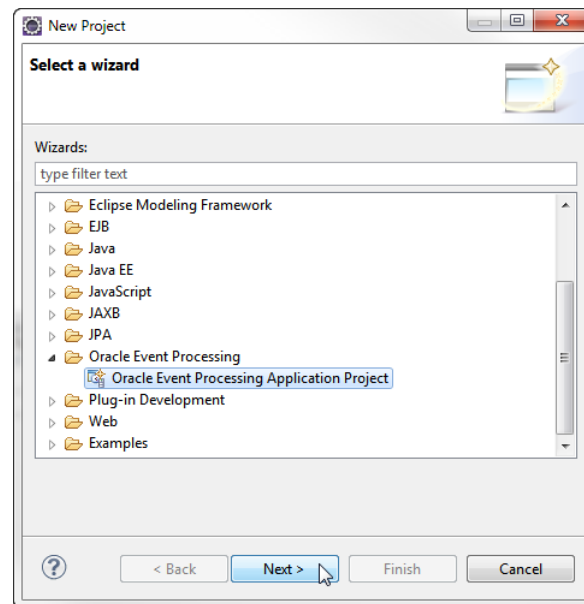
---

### To create an Oracle Event Processing project:

1. Open the EPN Editor (see [Section , "Opening the EPN Editor"](#))
2. Select **File > New Project**.

The New Project - Select a Wizard dialog appears as shown in [Figure 5-2](#).

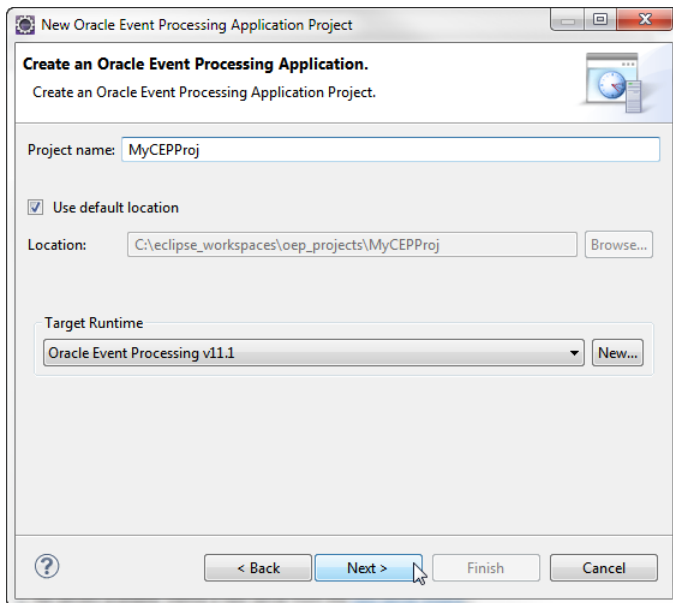
**Figure 5-2** *New Project - Select a Wizard Dialog*



3. Expand **Oracle Event Processing** and select **Oracle Event Processing Application Project**.
4. Click **Next**.

The New Oracle Event Processing Application Project wizard appears as shown in [Figure 5-3](#).

**Figure 5–3 New Oracle Event Processing Application Project Wizard: Create an Oracle Event Processing Application**



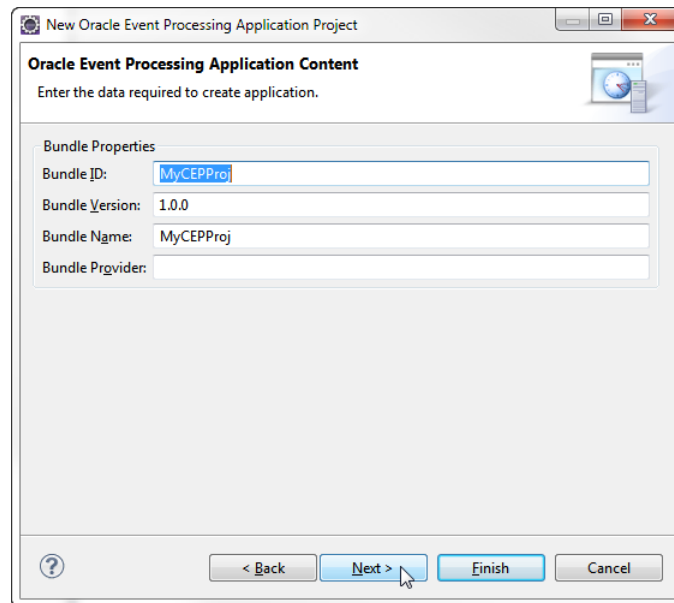
5. Configure the Create an Oracle Event Processing Application dialog as shown in [Table 5–2](#).

**Table 5–2 Create an Oracle Event Processing Application Dialog**

Attribute	Description
Project name	The name of your Oracle Event Processing project. This name will be used as the default name of your application when it is deployed to the Oracle Event Processing 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, clear <b>Use default location</b> and click <b>Browse</b> to place the project in a directory outside of your workspace.
Target Runtime	The Oracle Event Processing server you will deploy your project to.

6. Click **Next**.  
  
The Oracle Event Processing Application Content dialog appears as shown in [Figure 5–4](#).

**Figure 5–4 New Oracle Event Processing Application Project Wizard: Oracle Event Processing Application Content**



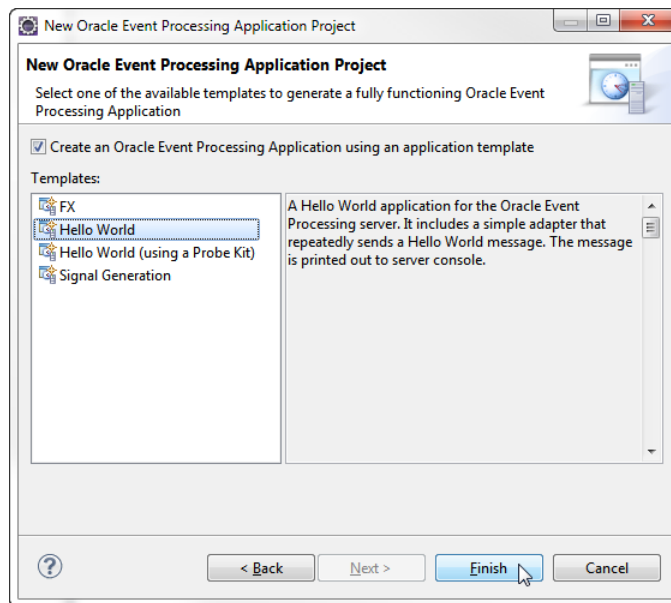
7. Optionally, configure the Oracle Event Processing Application Content dialog as shown in [Table 5–3](#).

**Table 5–3 Oracle Event Processing 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 5–5](#).

**Figure 5–5 New Oracle Event Processing Application Project Wizard: Template Dialog**

9. Optionally, select an Oracle Event Processing application template to pre-populate your project with the content that the template specifies.

10. Click **Finish**.

The Oracle Event Processing IDE for Eclipse creates the Oracle Event Processing project.

11. Optionally, create additional EPN assembly files and component configuration files.

By default, Oracle Event Processing IDE for Eclipse creates one EPN assembly file and one component configuration file for your project. Optionally, you may choose to define and configure Oracle Event Processing objects in multiple EPN assembly and component configuration files to improve management, concurrent development, and re-use.

For more information, see:

- [Section , "Creating EPN Assembly Files"](#)
- [Section , "Creating Component Configuration Files"](#)

## Creating EPN Assembly Files

You use the Event Processing Network (EPN) assembly file to declare the components that make up your Oracle Event Processing 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 [Section , "Foreign Exchange \(FX\) Example"](#). For additional information about Spring and OSGi, see [Appendix A, "Additional Information about Spring and OSGi."](#)



## How to Create a New EPN Assembly File Using Oracle Event Processing IDE for Eclipse

The most efficient and least error-prone way to create and edit the EPN file is using the New File Wizard in the Oracle Event Processing IDE for Eclipse.

For more information, see:

- [Section , "EPN Editor Overview"](#)
- [Section , "Managing Libraries and Other Non-Class Files in Oracle Event Processing Projects"](#)

### To create a new EPN assembly file using Oracle Event Processing IDE for Eclipse:

1. Create an Oracle Event Processing project.

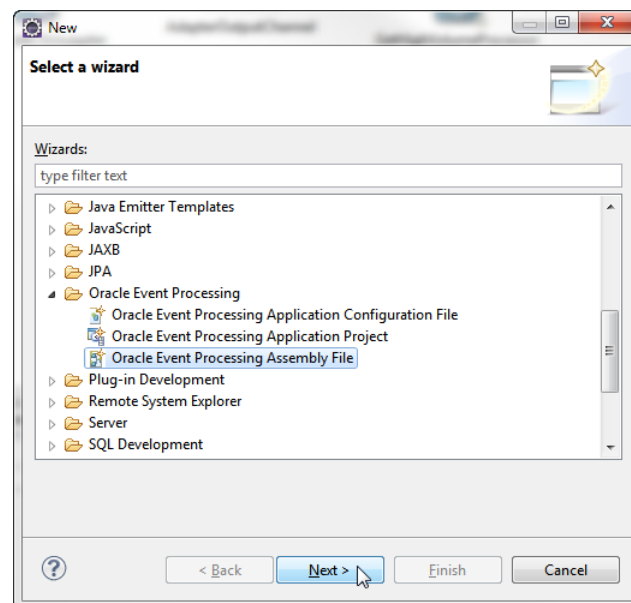
See [Section , "Creating Oracle Event Processing Projects"](#).

By default, Oracle Event Processing IDE for Eclipse creates one EPN assembly file for your project. Optionally, you may choose to define Oracle Event Processing objects in multiple EPN assembly files to improve management, concurrent development, and re-use.

2. To optionally create additional EPN assembly files:
  - a. Select **File > New > Other**.

The New dialog appears as [Figure 5–6](#) shows.

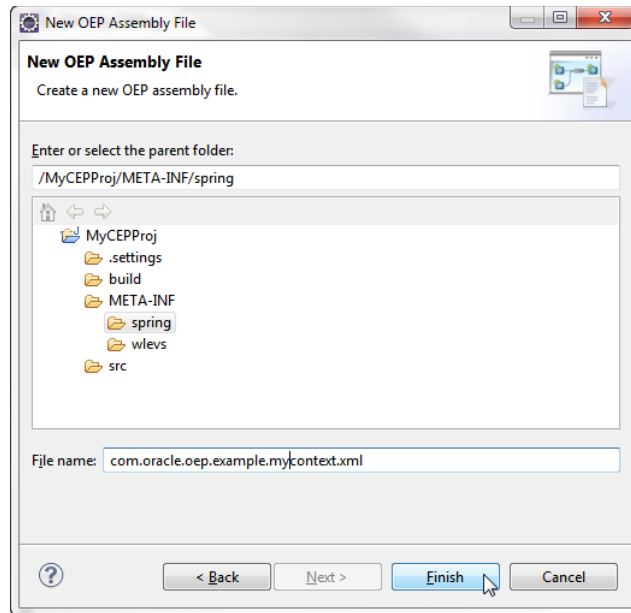
**Figure 5–6** New Dialog



- b. Expand **Oracle Event Processing** and select **Oracle Event Processing Assembly File**.
  - c. Click **Next**.

The New OEP Assembly File dialog appears as [Figure 5–7](#) shows.

**Figure 5–7 New OEP Assembly File Dialog**



- d. Configure the New OEP Assembly File dialog as shown in [Table 5–4](#).

**Table 5–4 New OEP Assembly File Dialog**

Attribute	Description
Enter or select the parent folder	Enter the fully qualified path to the folder in which Oracle Event Processing IDE for Eclipse will create the EPN assembly file or use the file system navigation controls to select the folder.
File name	Enter the name of the new EPN assembly file.

- e. Click **Finish**.
3. Open the EPN editor.  
See [Section , "Opening the EPN Editor"](#).
  4. If you created multiple EPN assembly files, you can select the EPN assembly file you want to work on using the EPN Editor **Filter** pull-down menu.  
  
This lets you focus on just the subset of the EPN that the selected EPN assembly file represents.  
  
To see the union of all components in all EPN assembly files, select **Full EPN** from the EPN Editor **Filter** pull-down menu.  
  
For more information, see [Section , "Filtering"](#).
  5. Create and connect nodes on the EPN.  
See [Section , "Using the EPN Editor"](#).

## Creating Component Configuration Files

You use a component configuration file to configure the components that make up your Oracle Event Processing application.

For an example of a component configuration file, see the [Section , "Foreign Exchange \(FX\) Example"](#).

## How to Create a New Component Configuration File Using Oracle Event Processing IDE for Eclipse

The most efficient and least error-prone way to create and edit a component configuration file is using the New File Wizard in the Oracle Event Processing IDE for Eclipse.

For more information, see:

- [Section , "EPN Editor Overview"](#)
- [Section , "Managing Libraries and Other Non-Class Files in Oracle Event Processing Projects"](#)

### To create a new component configuration file using Oracle Event Processing IDE for Eclipse:

1. Create an Oracle Event Processing project.

See [Section , "Creating Oracle Event Processing Projects"](#).

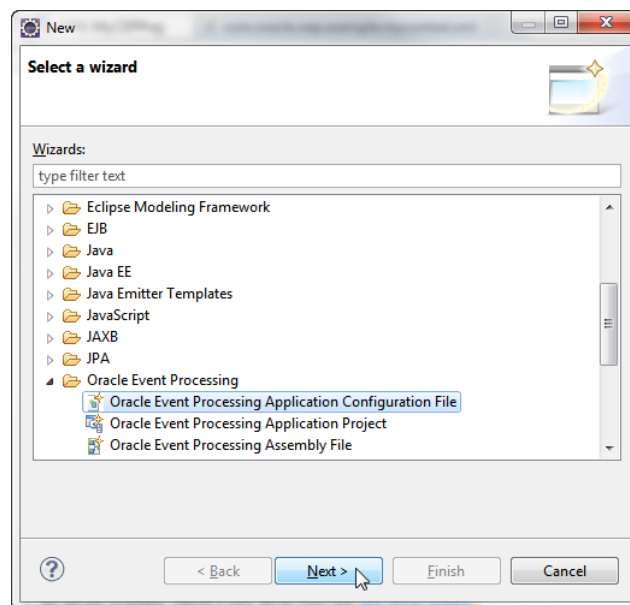
By default, Oracle Event Processing IDE for Eclipse creates one component configuration file for your project. Optionally, you may choose to configure Oracle Event Processing objects in multiple component configuration files to improve management, concurrent development, and re-use.

2. To optionally create component configuration files:

- Select **File > New > Other**.

The New dialog appears as [Figure 5–8](#) shows.

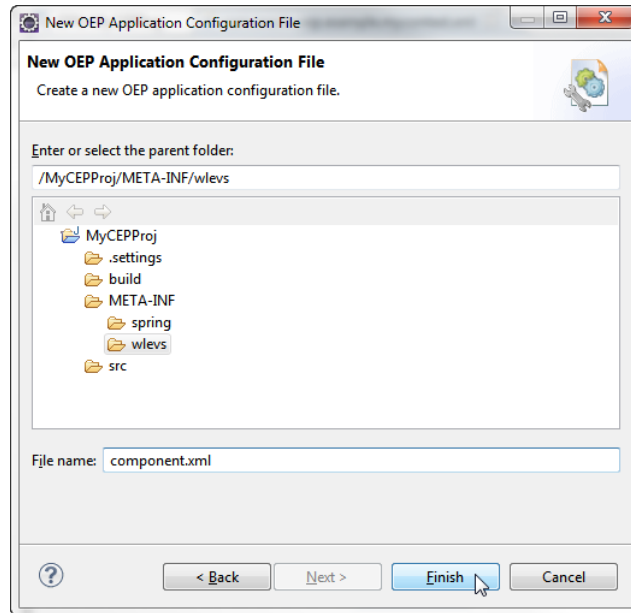
**Figure 5–8** New Dialog



- Expand **Oracle Event Processing** and select **Oracle Event Processing Application Configuration File**.

- Click **Next**.  
The New OEP Application Configuration File dialog appears as [Figure 5–9](#) shows.

**Figure 5–9 New OEP Application Configuration File Dialog**



- Configure the New OEP Assembly File dialog as shown in [Table 5–4](#).

**Table 5–5 New OEP Configuration File Dialog**

Attribute	Description
Enter or select the parent folder	Enter the fully qualified path to the folder in which Oracle Event Processing IDE for Eclipse will create the component configuration file or use the file system navigation controls to select the folder.
File name	Enter the name of the new component configuration file.

- Click **Finish**.
3. Open the EPN editor.  
See [Section , "Opening the EPN Editor"](#).
  4. Create and connect nodes on the EPN.  
See [Section , "Using the EPN Editor"](#).

## Exporting Oracle Event Processing Projects

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

## How to Export an Oracle Event Processing Project

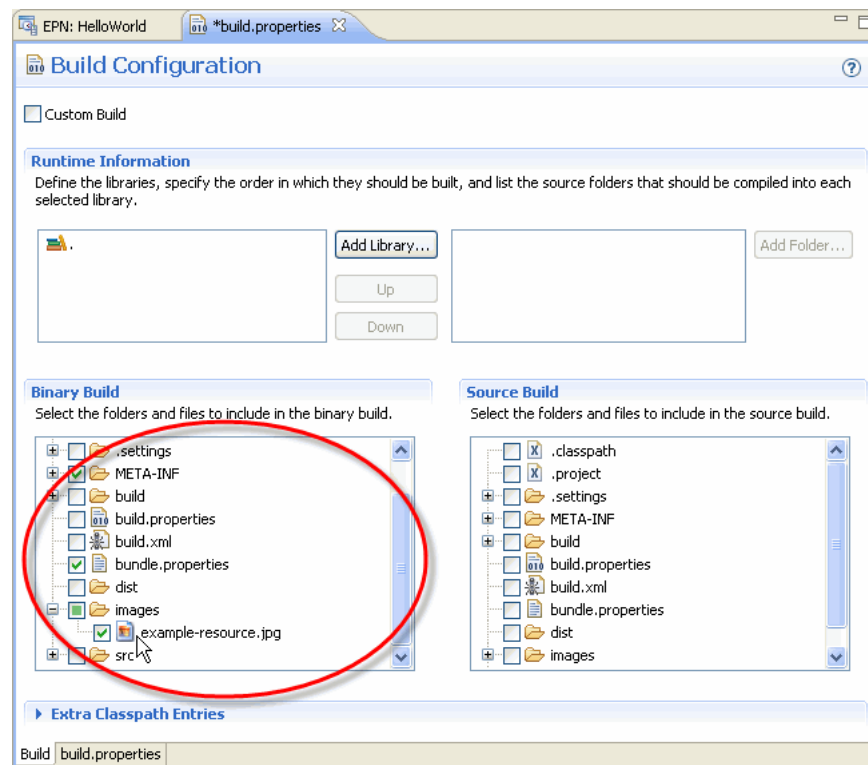
This section describes how to export an Oracle Event Processing project into an OSGi bundle.

### To export an Oracle Event Processing project:

1. Start the Oracle Event Processing IDE for Eclipse and open your Oracle Event Processing project.
2. The Oracle Event Processing 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.

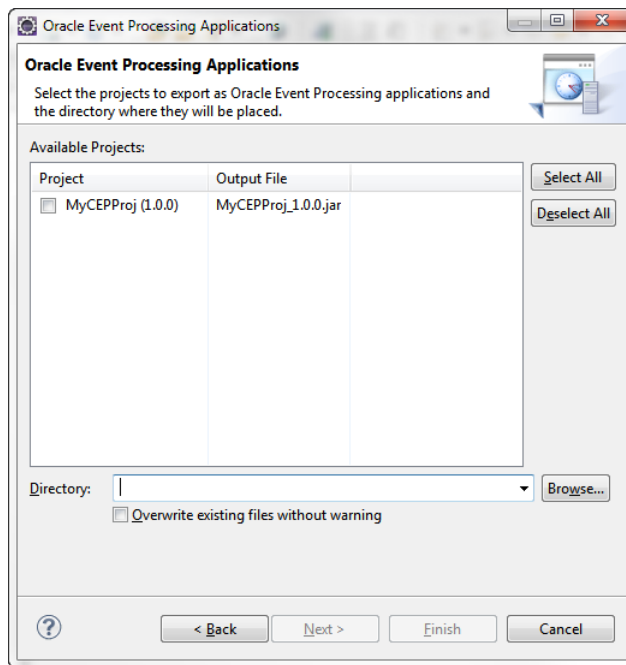
The `build.properties` file opens as shown in [Figure 5–10](#).

**Figure 5–10** Oracle Event Processing 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.
4. Expand the **Oracle Event Processing** option and select **Oracle Event Processing Applications**.
5. Click **Next**.  
The Oracle Event Processing Applications Export: Select Project dialog appears as shown in [Figure 5–11](#).

**Figure 5–11 Oracle Event Processing Applications Export: Select Project Dialog**



6. Configure the Oracle Event Processing Applications Export: Select Project dialog as shown in [Table 5–6](#).

**Table 5–6 Oracle Event Processing Application Content Dialog**

Attribute	Description
Available Projects	The list of Oracle Event Processing 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 Event Processing project JAR files are exported. Click <b>Browse</b> 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 Event Processing server.
  - a. If your JAR is an application, see [Section , "How to Deploy an Application to an Oracle Event Processing Server"](#).
  - b. If your JAR is an application library, see [Section , "Application Libraries"](#)
9. Deploy other dependent resources, if any, to your Oracle Event Processing server.  
For example:

- Other OSGi bundles your application depends on.  
Deploy these bundles on the Oracle Event Processing server using the Oracle Event Processing 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.

## Upgrading Projects

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

This section describes:

- [Section , "How to Upgrade Projects from Oracle Event Processing 2.1 to 10.3"](#)
- [Section , "How to Upgrade Projects from Oracle Event Processing 10.3 to 11g Release 1 \(11.1.1\)"](#)

For more information, see:

- [Section , "Configuring Eclipse"](#)
- *Oracle Fusion Middleware Getting Started Guide for Oracle Event Processing*

## How to Upgrade Projects from Oracle Event Processing 2.1 to 10.3

While project structure has stayed the same since 2.1, the data stored in Oracle Event Processing 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 Event Processing 2.1 to 10.3:

1. Open your Oracle Event Processing 2.1 project in Oracle Event Processing IDE for Eclipse.
2. Select **File > Switch Workspace > Other**.  
The Workspace Launcher dialog appears.

---

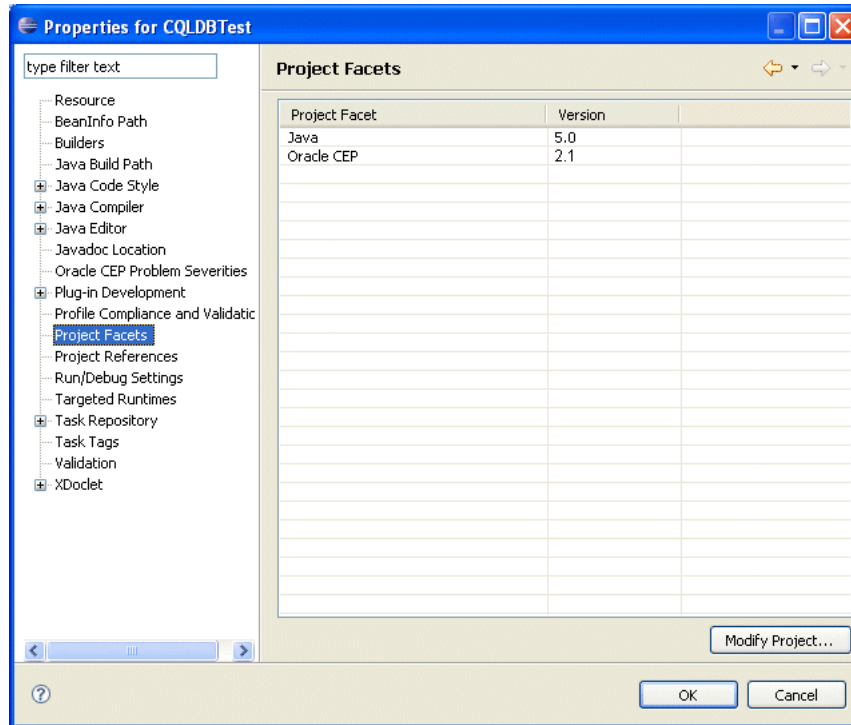
**Note:** Do not choose to copy settings from the current workspace.

---

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.
5. Expand the **General** option and select **Existing Projects into Workspace**.
6. Click **Next**.  
The Import Projects dialog appears.
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.

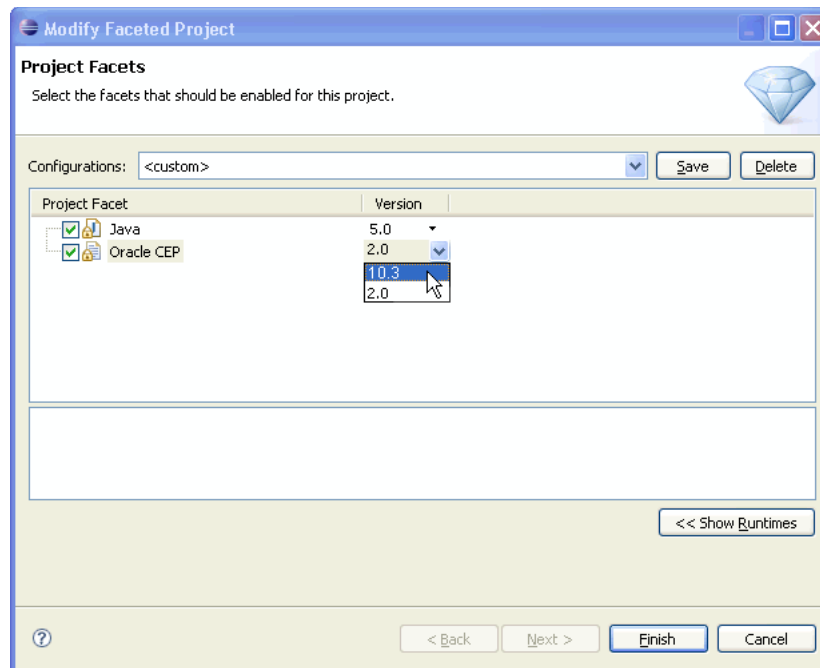
- 8. For each project, change the project facet version as follows:
  - a. Right-click your project and select **Properties**.  
 The Project Properties dialog appears as shown in [Figure 5–12](#).

**Figure 5–12 Project Properties Dialog: Project Facets**



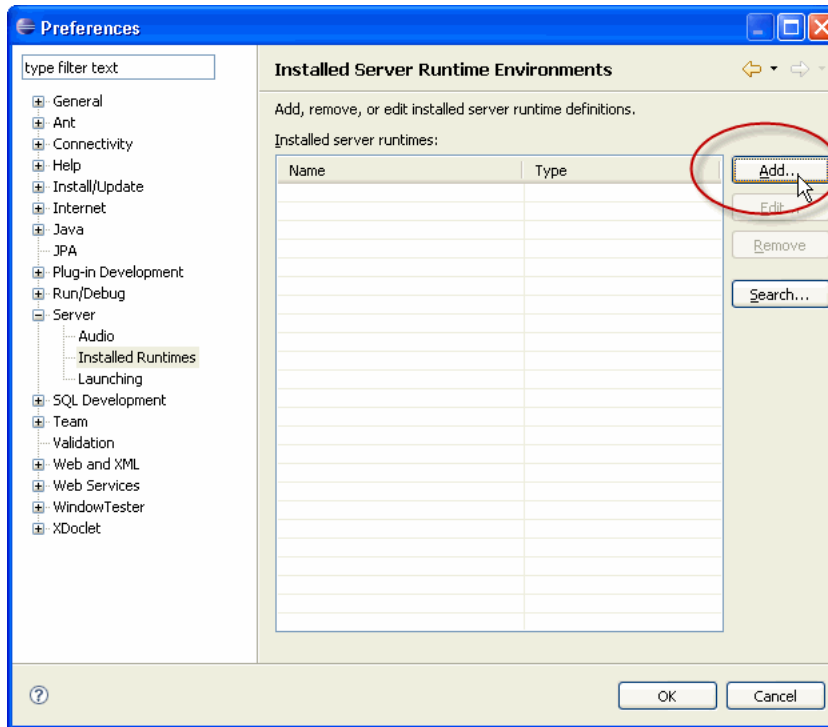
- b. Select the Project Facets option.  
 The Project Facet properties are displayed as [Figure 5–12](#) shows.
- c. Click **Modify Project**.  
 The Modify Faceted Project dialog appears shown in [Figure 5–13](#).



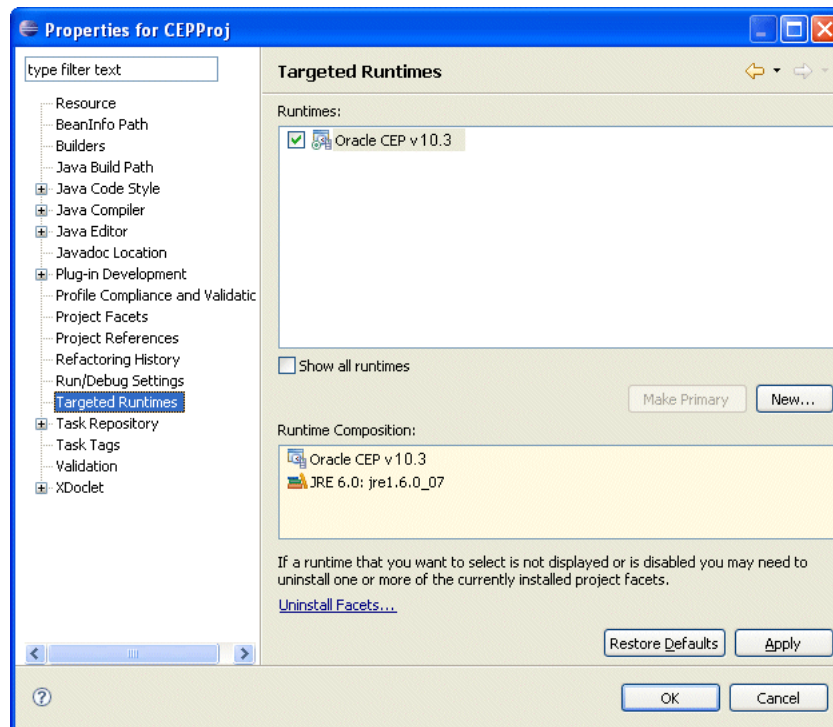
**Figure 5–13 Modify Faceted Project**

- d. For the Oracle Event Processing 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 Event Processing server runtime:
    - a. Select **Window > Preferences**.  
The Preferences dialog appears as shown in [Figure 5–14](#).

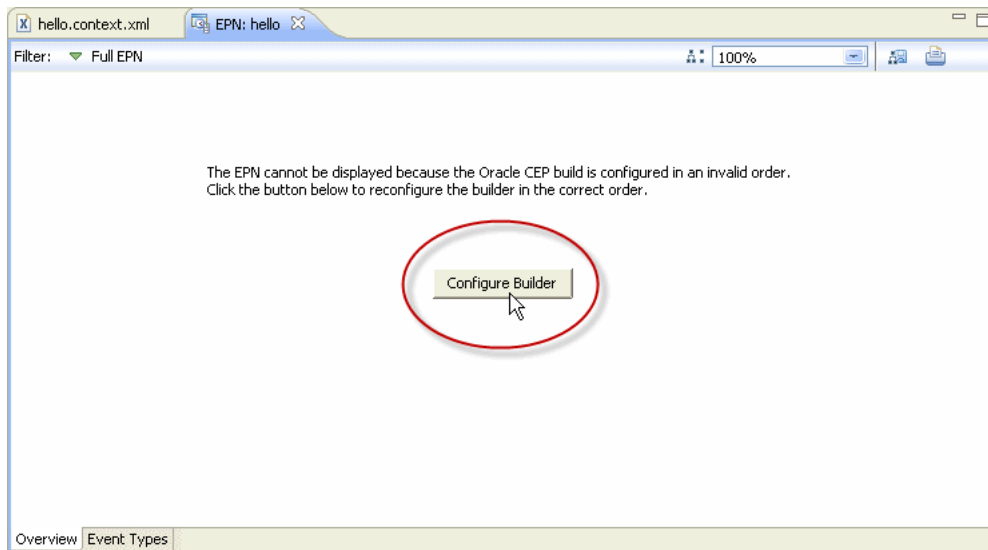
**Figure 5–14 Preferences Dialog**



- b.** Expand the **Server** option and select **Installed Runtimes**.
  - c.** Add a new 10.3 Oracle Event Processing server runtime as [Section , "How to Create an Oracle Event Processing Server Runtime"](#) describes.
  - d.** Click **OK**.
- 10.** For each project, specify the new 10.3 Oracle Event Processing server runtime you created:
  - a.** Right-click your project and select **Properties**.  
The Project Properties dialog appears as shown in [Figure 5–15](#).

**Figure 5–15 Project Properties Dialog: Targeted Runtimes**

- b. Select the **Targeted Runtimes** option.  
The Targeted Runtimes properties are displayed as [Figure 5–15](#) shows.
  - c. Check the new Oracle Event Processing 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 5–16](#), click the **Configure Builder** button.

**Figure 5–16 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 Event Processing 10.3" in the *Oracle Fusion Middleware Getting Started Guide for Oracle Event Processing*
- *Oracle Event Processing Release Notes for 10.3*  
([http://download.oracle.com/docs/cd/E13157\\_01/wlevs/docs30/notes/notes.html](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.

## How to Upgrade Projects from Oracle Event Processing 10.3 to 11g Release 1 (11.1.1)

While project structure has stayed the same since 10.3, the data stored in Oracle Event Processing Projects has changed significantly. It is therefore necessary to take steps to upgrade 10.3 projects manually before continuing their development in 11g Release 1 (11.1.1).

The following outlines the steps necessary to upgrade 10.3 projects to 11g Release 1 (11.1.1)

**To upgrade projects from Oracle Event Processing 10.3 to 11g Release 1 (11.1.1)**

1. Open your Oracle Event Processing 10.3 project in Oracle Event Processing IDE for Eclipse.
2. Select **File > Switch Workspace > Other**.  
The Workspace Launcher dialog appears.
3. Click Browse and select a new workspace directory.

---

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

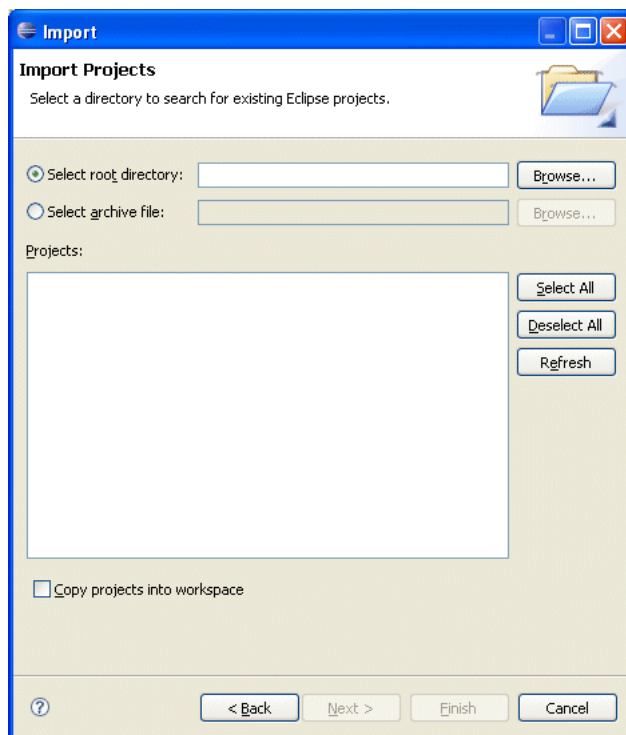
---

Eclipse exits and restarts using the new workspace.

4. Select **File > Import**.  
The Import Dialog appears.
5. Expand the **General** option and select **Existing Projects into Workspace**.
6. Click **Next**.

The Import Projects dialog appears as shown in [Figure 5–17](#).

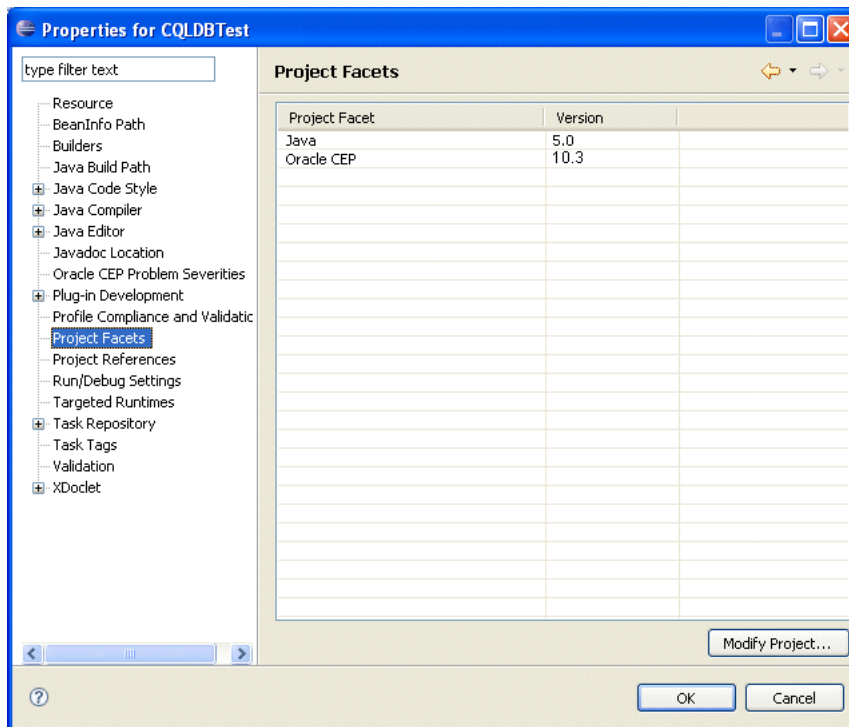
**Figure 5–17 Import Projects Dialog**



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 5–12](#).

**Figure 5–18 Project Properties Dialog: Project Facets**



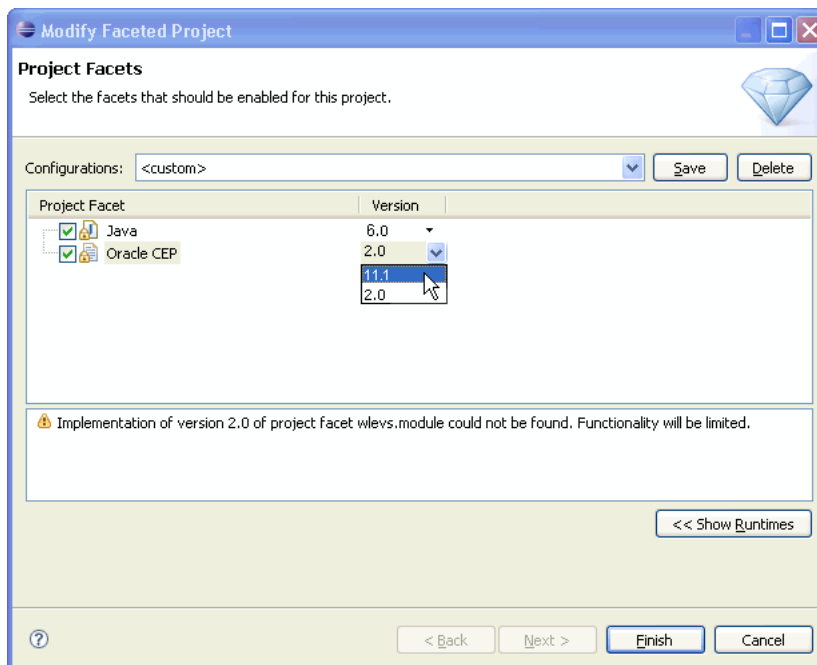
**b.** Select the Project Facets option.

The Project Facet properties are displayed as [Figure 5–12](#) shows.

**c.** Click **Modify Project**.

The Modify Faceted Project dialog appears shown in [Figure 5–13](#).

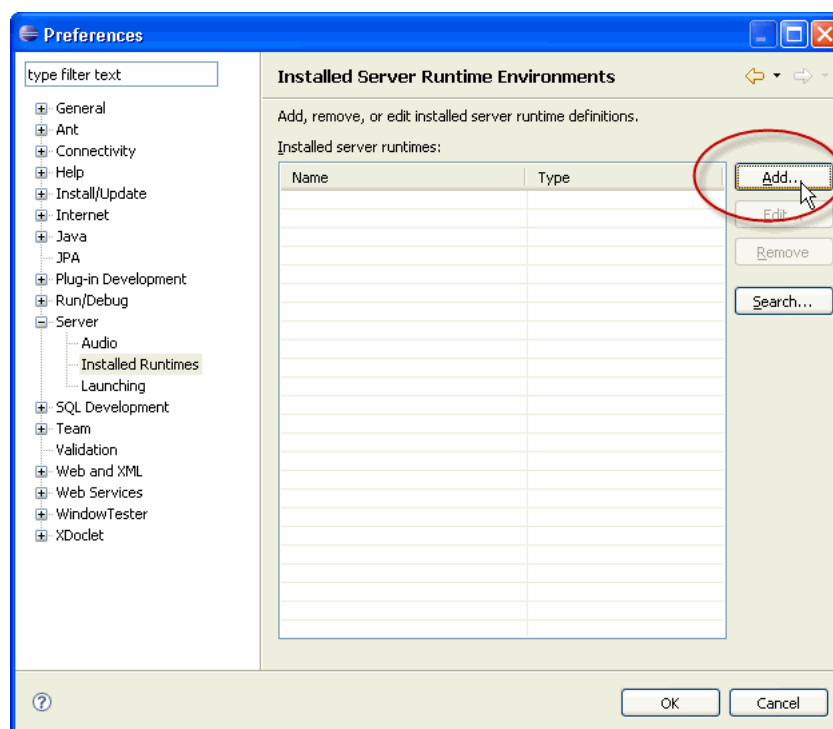
**Figure 5–19 Modify Faceted Project**



- d. For the Java facet, select 6.0 from the **Version** pull-down menu.
  - e. For the Oracle Event Processing 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 Event Processing server runtime:
    - a. Select **Window > Preferences**.

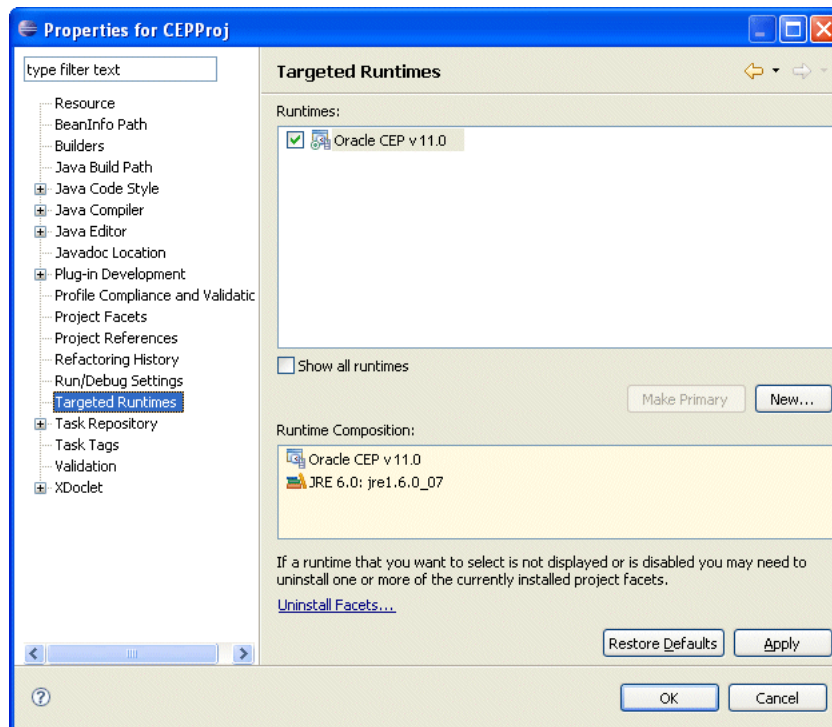
The Preferences dialog appears as shown in [Figure 5–14](#).

**Figure 5–20 Preferences Dialog**



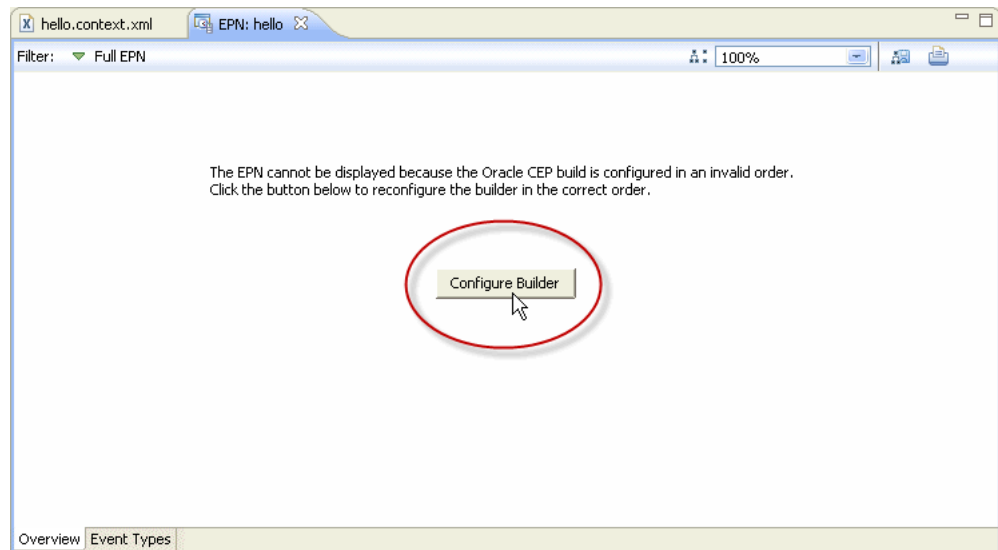
- b. Expand the **Server** option and select **Installed Runtimes**.
  - c. Add a new 11.0 Oracle Event Processing server runtime as [Section , "How to Create an Oracle Event Processing Server Runtime"](#) describes.
  - d. Click **OK**.
10. For each project, specify the new 11.0 Oracle Event Processing server runtime you created:
    - a. Right-click your project and select **Properties**.

The Project Properties dialog appears as shown in [Figure 5–15](#).

**Figure 5–21 Project Properties Dialog: Targeted Runtimes**

- b. Select the **Targeted Runtimes** option.  
The Targeted Runtimes properties are displayed as [Figure 5–15](#) shows.
  - c. Check the new Oracle Event Processing 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 5–16](#), click the **Configure Builder** button.



**Figure 5–22 Builder Error**

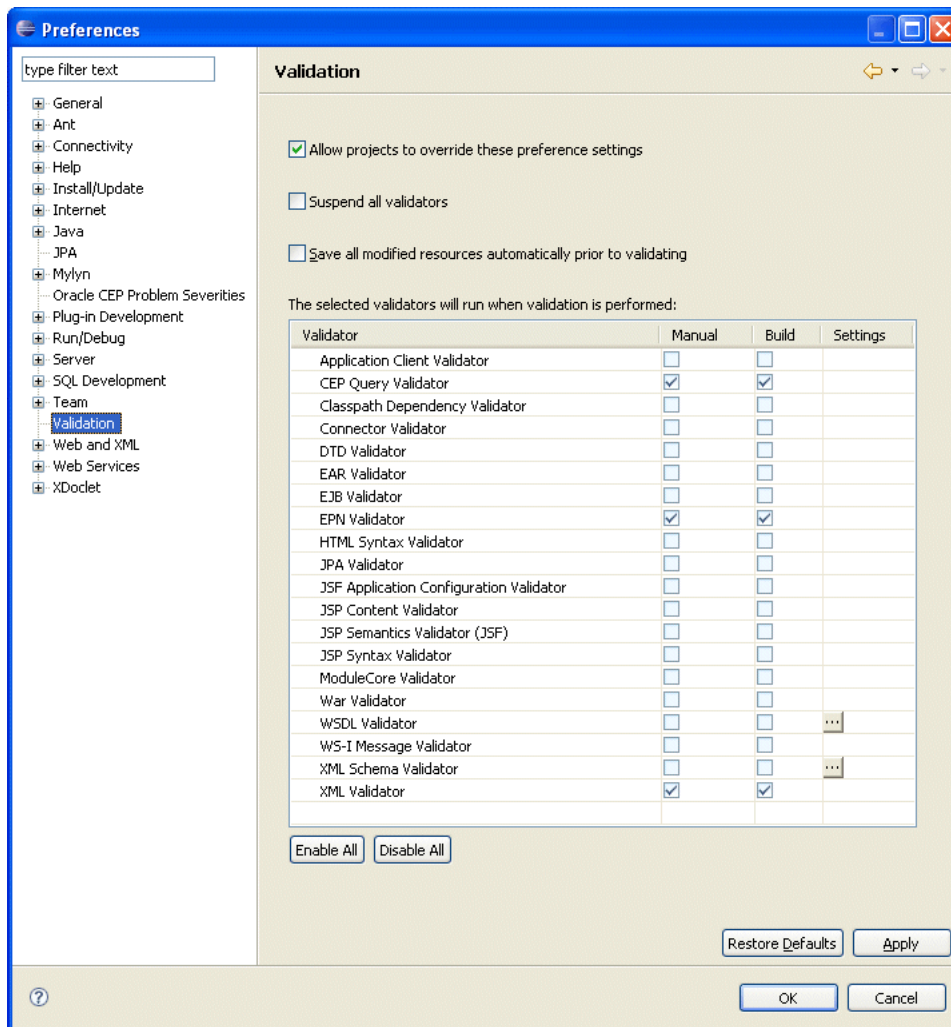
The EPN diagram is displayed.

d. Repeat for the next project.

12. Select **Window > Preferences**.

The Preferences dialog appears as shown in [Figure 5–23](#).

**Figure 5–23 Preferences Dialog**



13. Select the **Validation** option.

14. Ensure that the following validation options are checked:

- CQL Validator
- EPN Validator
- XML Validator

15. Clear 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 Event Processing 10.3 Application to Run on Oracle Event Processing Release 11gR1 (11.1.1)" in the *Oracle Fusion Middleware Getting Started Guide for Oracle Event Processing*

## Managing Libraries and Other Non-Class Files in Oracle Event Processing Projects

Many projects require the use of non-class files such as libraries or property files 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 add the following non-class files to an Oracle Event Processing project, 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 Event Processing 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 , "How to Add a Standard JAR File to an Oracle Event Processing Project"](#)
- [Section , "How to Add an OSGi Bundle to an Oracle Event Processing Project"](#)
- [Section , "How to Add a Property File to an Oracle Event Processing Project"](#)
- [Section , "How to Export a Package"](#)
- [Section , "How to Import a Package"](#)
- [Section , "Accessing Third-Party JAR Files Using -Xbootclasspath"](#)
- **OSGi Bundles:** If your library is already 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 `DOMAIN_DIR/servername/modules` directory of their machine, rather than have it source controlled with the rest of the project.

The main advantage of the OSGi bundle library option is that you can use the Oracle Event Processing server application library to manage OSGi bundle libraries to ensure that they are deployed before any applications that depend on them.

For more information, see:

- [Section , "How to Add an OSGi Bundle to an Oracle Event Processing Project"](#)
- [Section , "Application Libraries"](#)
- **Property Files:** Adding a Java property file to a project allows you to manage properties efficiently. You can add a Java property file to an Oracle Event

Processing project so that the property file is deployed with your application and is available at runtime.

For more information, see:

- [Section , "How to Add a Property File to an Oracle Event Processing Project"](#)
- [Section , "Accessing Component and Server Configuration Using the ConfigurationPropertyPlaceholderConfigurer Class"](#)

## How to Add a Standard JAR File to an Oracle Event Processing Project

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

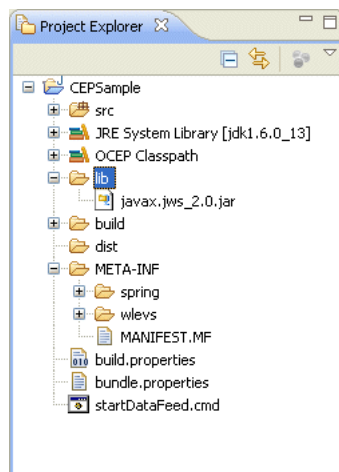
When you add a standard JAR file to an Oracle Event Processing project, you can optionally expose some or all of its packages to other bundles that will depend on this bundle.

### To add a standard JAR file to an Oracle Event Processing project:

1. Create a folder in your Oracle Event Processing IDE for Eclipse project to put the JAR file in.  
Oracle recommends that you create a folder to put them in such as `lib`.  
To create a new folder, right-click your project folder and select **New > Folder**.
2. Outside of the Oracle Event Processing IDE for Eclipse, copy your JAR file into the `lib` folder.
3. Inside the Oracle Event Processing IDE for Eclipse, right-click the `lib` folder and select **Refresh**.

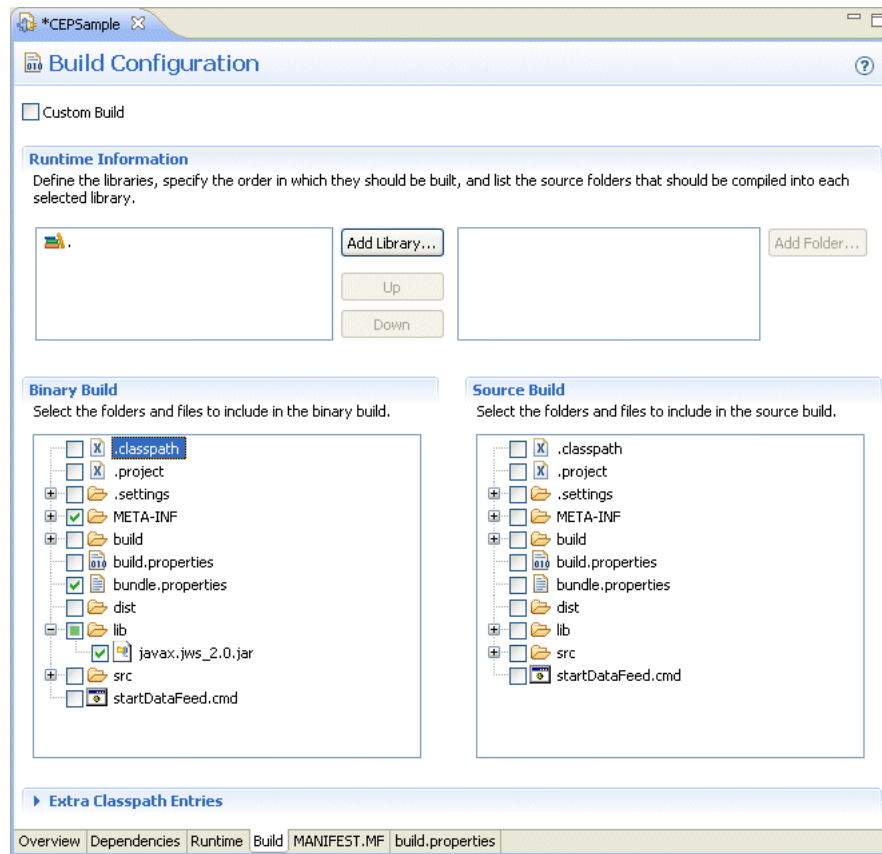
The JAR file appears in the `lib` folder as [Figure 5–24](#) shows.

**Figure 5–24 Oracle Event Processing IDE for Eclipse lib Directory**



4. Expand the `META-INF` directory and right-click the `MANIFEST.MF` file and select **Open With > Plug-in Manifest Editor**.

The Manifest Editor opens as [Figure 5–25](#) shows.

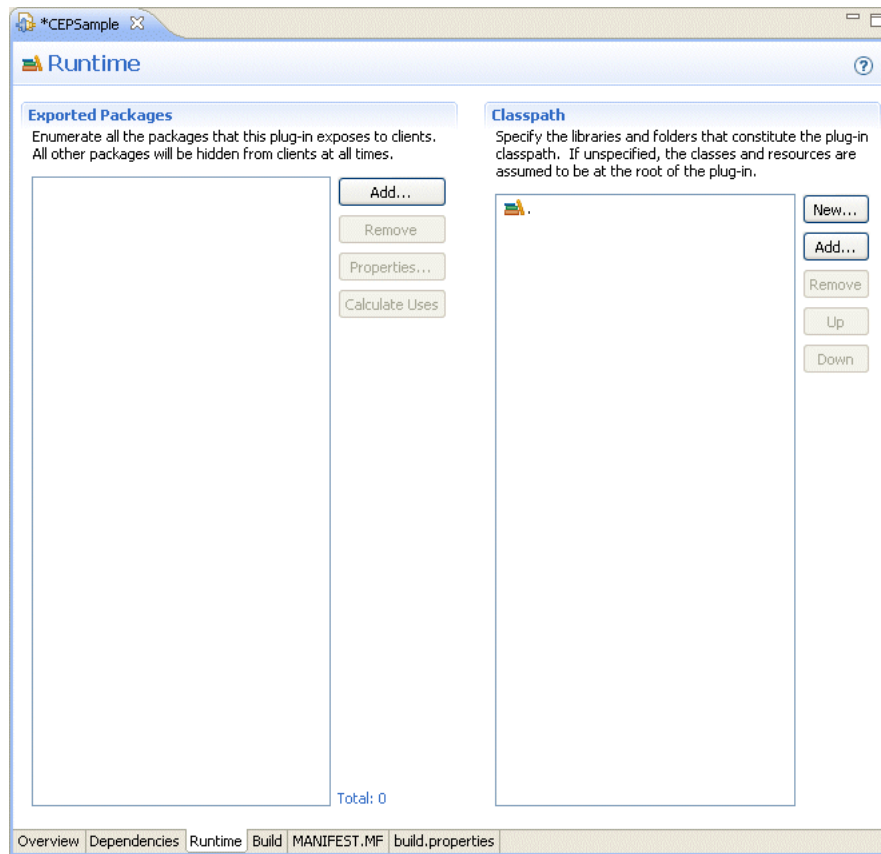
**Figure 5–25 Manifest Editor: Build Tab**

5. Click the **Build** tab.
6. Add your JAR file to the binary build under the project root as follows:
  - In the Binary Build area, expand the `lib` directory.
  - Check the box next to your library as [Figure 5–25](#) shows.
  - Press hit CTRL-SHIFT-S to save all files.

This edits the `build.properties` file in your project, and tells the Oracle Event Processing IDE for Eclipse to add the JAR file to your bundle when you build the bundle JAR.

7. In the Manifest Editor, click the **Runtime** tab.  
The Runtime tab appears as [Figure 5–26](#) shows.

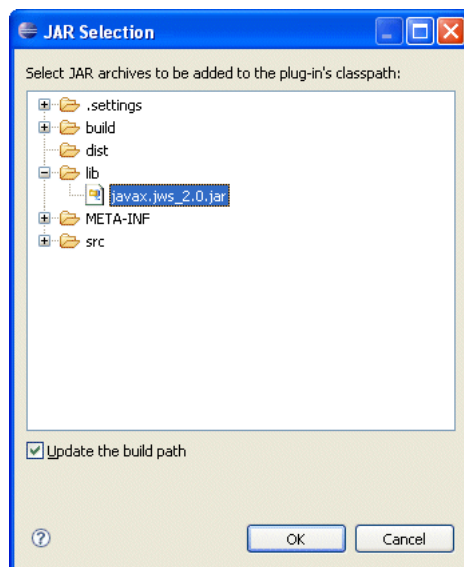
**Figure 5–26 Manifest Editor - Runtime Tab**



8. Add the JAR file to your project's classpath as follows:
  - In the Manifest Editor, click the **Add** button.

The JAR Selection dialog appears as shown in [Figure 5–27](#).

**Figure 5–27 JAR Selection Dialog**



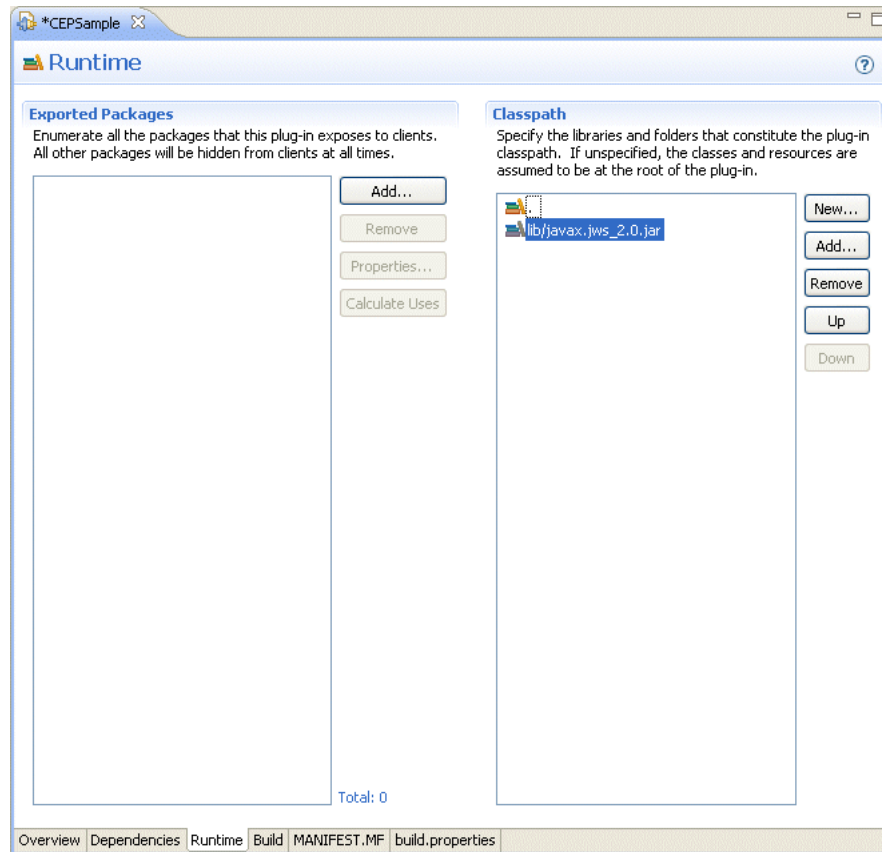
- Select the JAR you want to add to the bundle.

In this example, expand the **lib** directory and select the `javax.jws_2.0.jar` file.

- Click **OK**.

This adds the selected JAR to the Classpath list as [Figure 5–28](#) shows.

**Figure 5–28** Manifest Editor Runtime tab After Adding a JAR to the Classpath

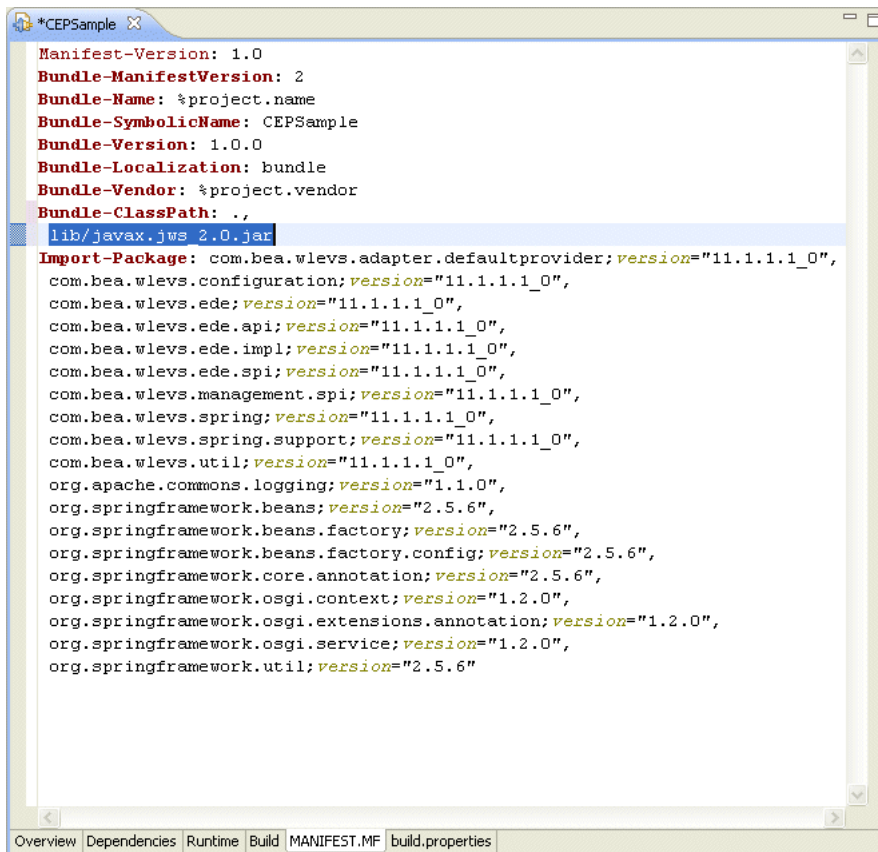


- Press hit **CTRL-SHIFT-S** to save all files.

This edits the `MANIFEST.MF` file, putting the JAR on your project classpath.

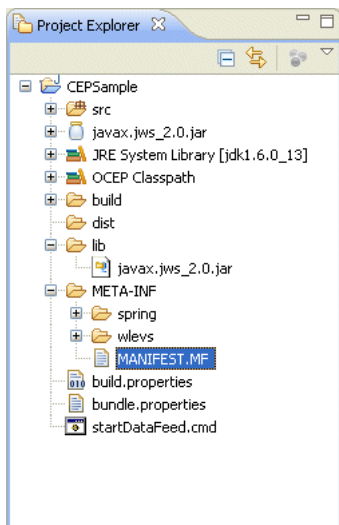
To confirm this, in the Manifest Editor, click the **MANIFEST.MF** tab and note that the JAR is now listed under the `Bundle-Classpath` property as [Figure 5–29](#) shows.

**Figure 5–29 Manifest Editor MANIFEST.MF Tab**



Note also that the JAR now appears as a library at the root of the project as [Figure 5–30](#) shows.

**Figure 5–30 Package Explorer**



- Optionally, if your bundle needs to export packages from this JAR to other bundles that will depend on this bundle, then you can export these packages as [Section , "How to Export a Package"](#) describes.



## How to Add an OSGi Bundle to an Oracle Event Processing Project

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

To add an OSGi bundle to an Oracle Event Processing project, you add the bundle to that bundle's dependencies definition.

---

**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 Event Processing server manually. For more information, see [Section , "Application Libraries"](#).

---

### To add an OSGi bundle to an Oracle Event Processing project:

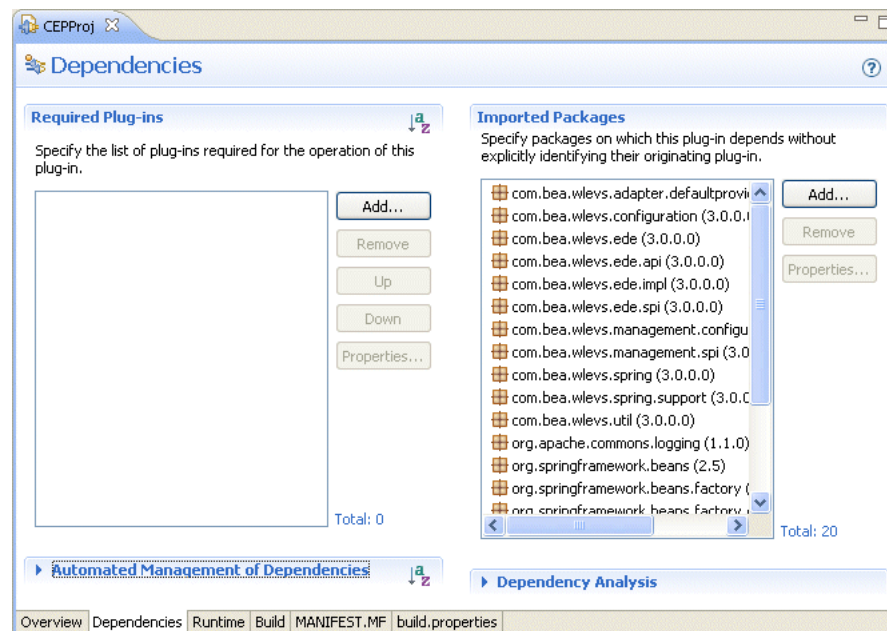
1. Place the OSGi bundle in the `DOMAIN_DIR/servername/modules` directory, where `DOMAIN_DIR` refers to the domain directory such as `/oracle_cep/user_projects/domains/mydomain` and `servername` refers to the server instance, such as `myserver`. For example:

```
c:\oracle_cep\user_projects\domains\mydomain\myserver\modules
```

2. Start the Oracle Event Processing IDE for Eclipse.
3. Right-click the project and select **Refresh Targeted Runtime**.
4. Right-click the `META-INF/MANIFEST.MF` file and select **Open With > Plug-in Manifest Editor**.

The Manifest Editor opens as [Figure 5–31](#) shows.

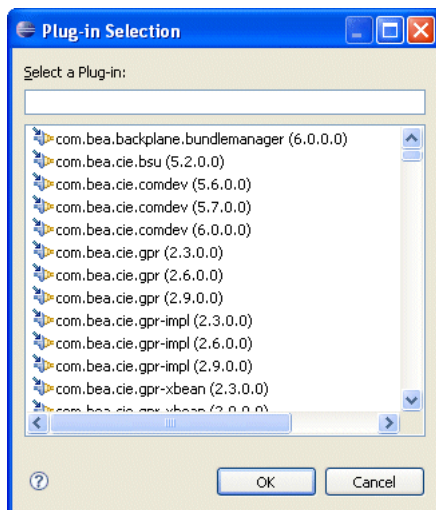
**Figure 5–31 Manifest Editor: Dependencies Tab**



5. Click the **Dependencies** tab.
6. In the **Required Plug-ins** area, click **Add**.

The Plug-in Selection dialog appears as shown in [Figure 5–32](#).

**Figure 5–32 Plug-in Selection Dialog**



7. Select the bundle you added to the `DOMAIN_DIR/servername/modules` directory of your Oracle Event Processing server installation directory in step 1 and click **OK**.

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

## How to Add a Property File to an Oracle Event Processing Project

You can add a Java property file to an Oracle Event Processing project so that the property file is deployed with your application and is available at runtime.

### To add a property file to an Oracle Event Processing project:

1. Create a folder in your Oracle Event Processing IDE for Eclipse project to put the property files in.

Oracle recommends that you create a folder to put them in such as `properties`.

To create a new folder, tight-click your project folder and select **New > Folder**.

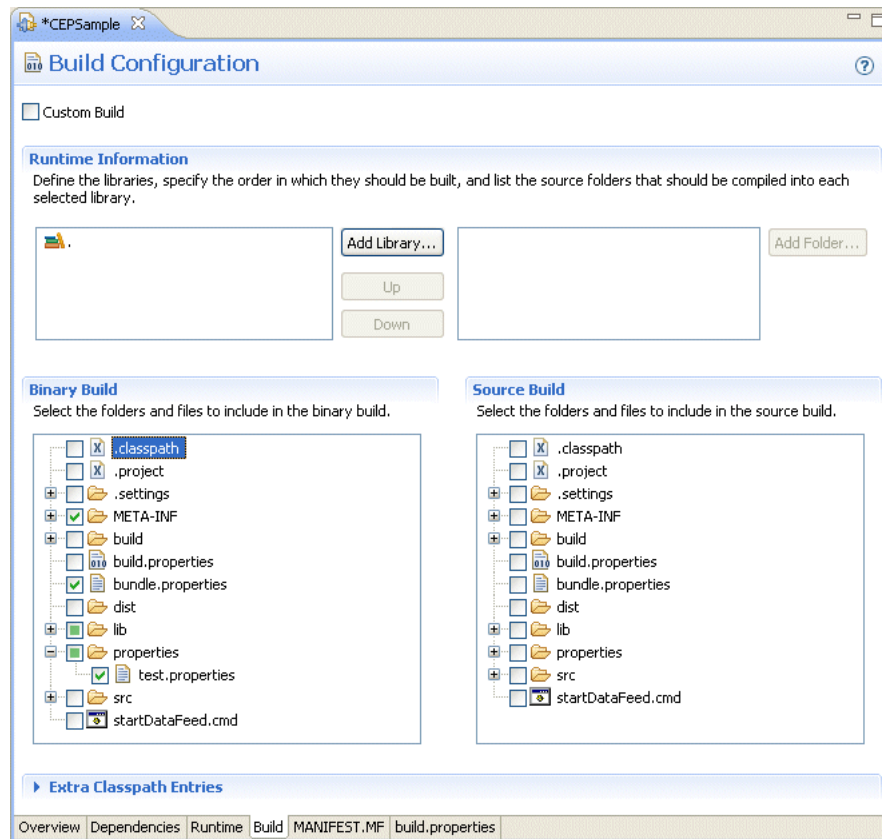
2. Outside of the Oracle Event Processing IDE for Eclipse, copy your property file into the `properties` folder.

3. Inside the Oracle Event Processing IDE for Eclipse, right-click the `properties` folder and select **Refresh**.

The property file appears in the `properties` folder.

4. Expand the `META-INF` directory and right-click the `MANIFEST.MF` file and select **Open With > Plug-in Manifest Editor**.

The Manifest Editor opens as [Figure 5–33](#) shows.

**Figure 5–33 Manifest Editor: Build Tab**

5. Click the **Build** tab.
6. Add your property file to the binary build under the project root as follows:
  - In the Binary Build area, expand the `properties` directory.
  - Check the box next to your property file as [Figure 5–33](#) shows.
  - Press hit CTRL-SHIFT-S to save all files.

This edits the `build.properties` file in your project, and tells the Oracle Event Processing IDE for Eclipse to add the property file to your bundle when you build the bundle JAR.

7. You can access the properties file in Java code as [Example 5–1](#) shows:

#### **Example 5–1 Accessing a Properties File**

```
public void onInsertEvent(Object event) {
    if (event instanceof HelloWorldEvent) {
        HelloWorldEvent helloWorldEvent = (HelloWorldEvent) event;
        InputStream resourceAsStream = getClass().getClassLoader().getResourceAsStream(
            "properties/test.properties"
        );
        Properties props = new Properties();
        try {
            props.load(resourceAsStream);
        } catch (IOException e) {
            e.printStackTrace();
        }
        System.out.println("Message: " + props.get("test-key"));
        // Throw com.bea.wllevs.ede.api.EventRejectedException to have exceptions propagated
    }
}
```

```

        // up to senders. Other errors will be logged and dropped.
    }
}

```

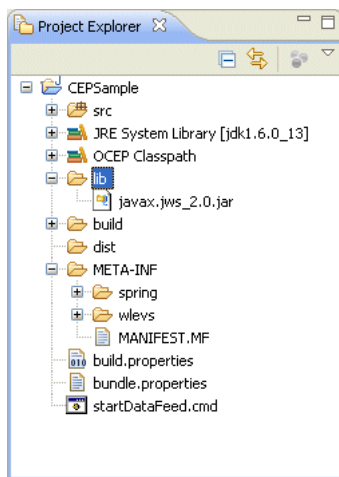
## How to Export a Package

Optionally, if your bundle needs to export a package from a JAR to other bundles that will depend on this bundle, then you can export this package. By doing so, you update the `Package-Export` MANIFEST entry to create an OSGi exporter for the package.

### To export a package:

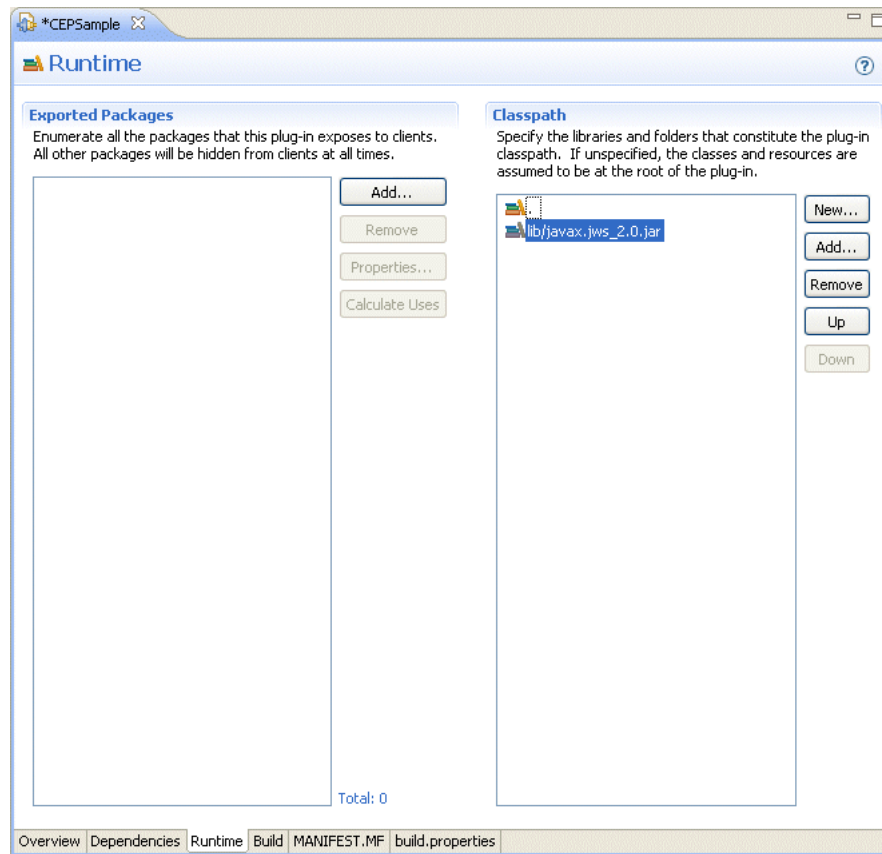
1. Inside the Oracle Event Processing IDE for Eclipse, expand the `META-INF` directory as [Figure 5-34](#) shows.

**Figure 5-34 Oracle Event Processing IDE for Eclipse lib Directory**



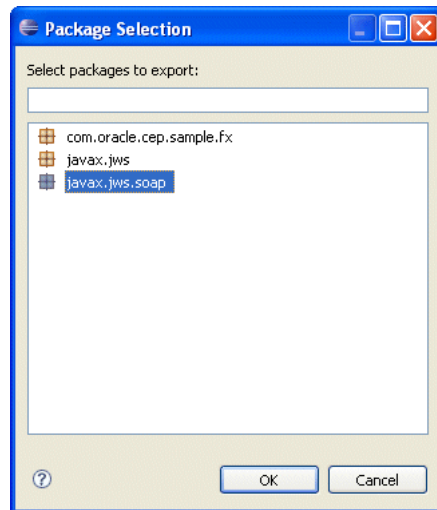
2. Right-click the `MANIFEST.MF` file and select **Open With > Plug-in Manifest Editor**. The Manifest Editor opens as [Figure 5-35](#) shows.

**Figure 5–35 Manifest Editor: Runtime tab**



3. Click the **Runtime** tab.  
The Runtime tab appears as [Figure 5–35](#) shows.
4. In the Exported Packages area, click the **Add** button.  
The Package Selection dialog appears as [Figure 5–36](#) shows.

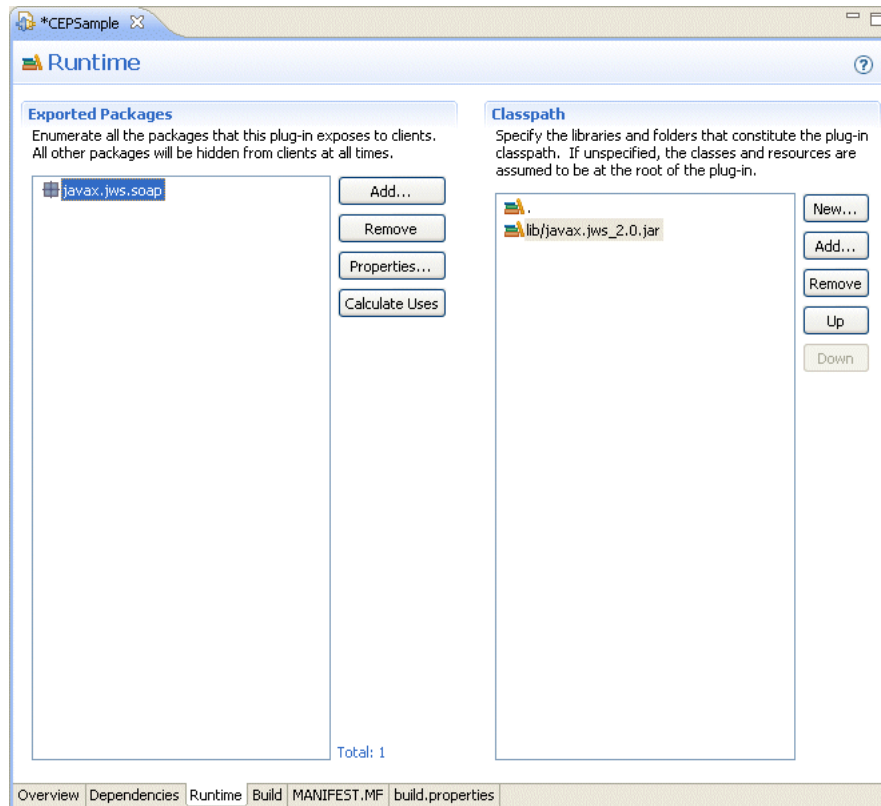
**Figure 5–36 Package Selection Dialog**



5. Select the package you want to export.

- To find a package in the list by name, type the name into the text field.
- In this example, select the `javax.jws.soap` package.
- 6. Click **OK**.
- The selected package is added to the Exported Packages area as [Figure 5–37](#) shows.

**Figure 5–37 Manifest Editor Runtime tab After Exporting a Package**



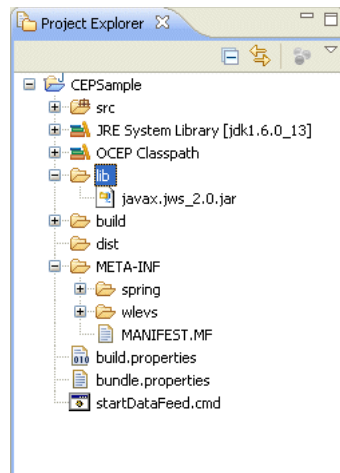
- 7. Press CTRL-SHIFT-S to save all files.

## How to Import a Package

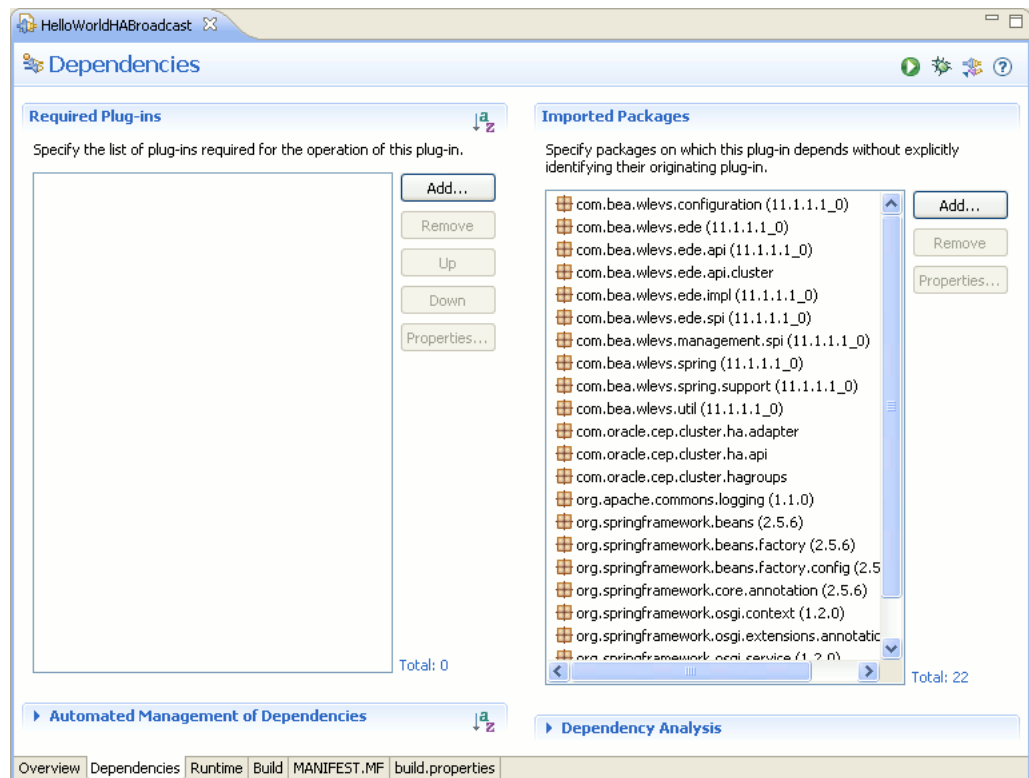
Optionally, if your bundle needs to import a package from a JAR, then you can import this package. By doing so, you update the Package-Import MANIFEST entry to create an OSGi importer for the package.

### To import a package:

- 1. Inside the Oracle Event Processing IDE for Eclipse, expand the META-INF directory as [Figure 5–38](#) shows.

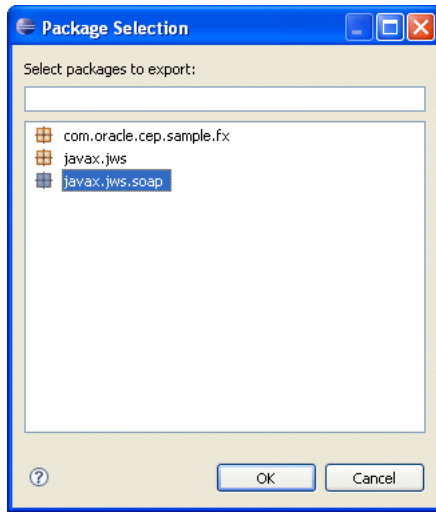
**Figure 5–38 Oracle Event Processing IDE for Eclipse lib Directory**


- Right-click the `MANIFEST.MF` file and select **Open With > Plug-in Manifest Editor**. The Manifest Editor opens as [Figure 5–39](#) shows.

**Figure 5–39 Manifest Editor: Dependencies tab**


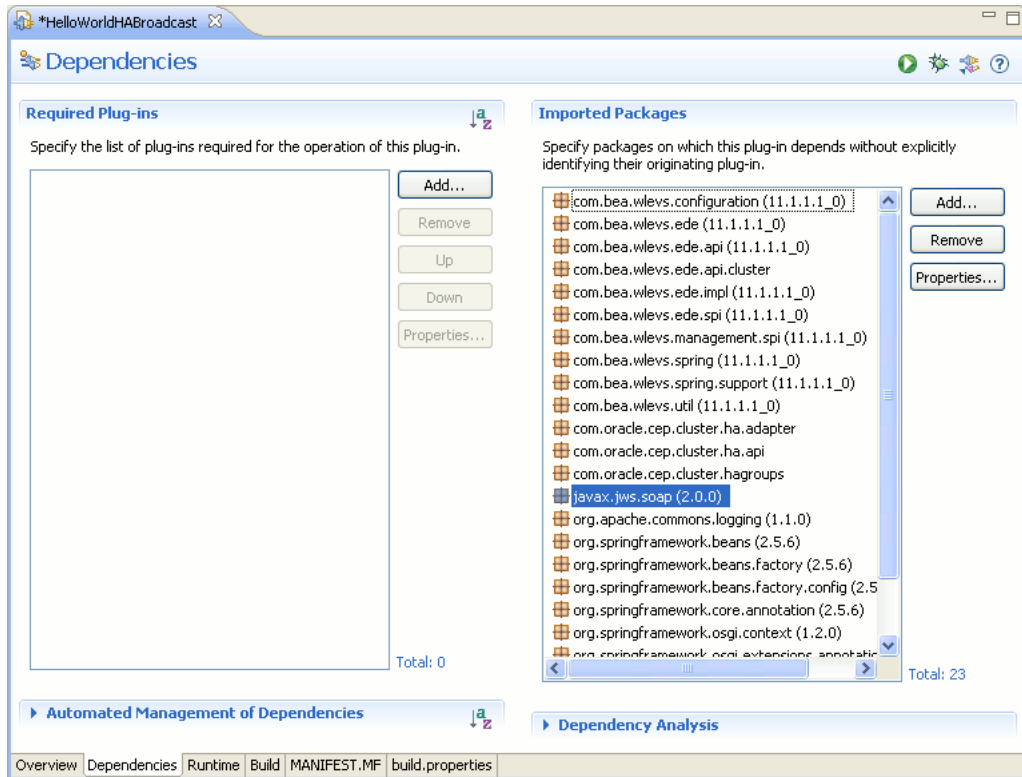
- Click the **Dependencies** tab. The Dependencies tab appears as [Figure 5–39](#) shows.
- In the Imported Packages area, click the **Add** button. The Package Selection dialog appears as [Figure 5–40](#) shows.

**Figure 5–40 Package Selection Dialog**



5. Select the package you want to import.  
 To find a package in the list by name, type the name into the text field.  
 In this example, select the `javax.jws.soap` package.
6. Click **OK**.  
 The selected package is added to the Import Packages area as [Figure 5–41](#) shows.

**Figure 5–41 Manifest Editor Dependencies tab After Importing a Package**



7. Press **CTRL-SHIFT-S** to save all files.



## Configuring Oracle Event Processing IDE for Eclipse Preferences

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

- [Section , "How to Configure Application Library Path Preferences"](#)
- [Section , "How to Configure Problem Severity Preferences"](#)

### How to Configure Application Library Path Preferences

You can define the path to an Oracle Event Processing server domain directory that contains application libraries that extend the Oracle Event Processing runtime.

For more information, see [Section , "How to Define the Application Library Directory Using Oracle Event Processing IDE for Eclipse"](#).

### How to Configure Problem Severity Preferences

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

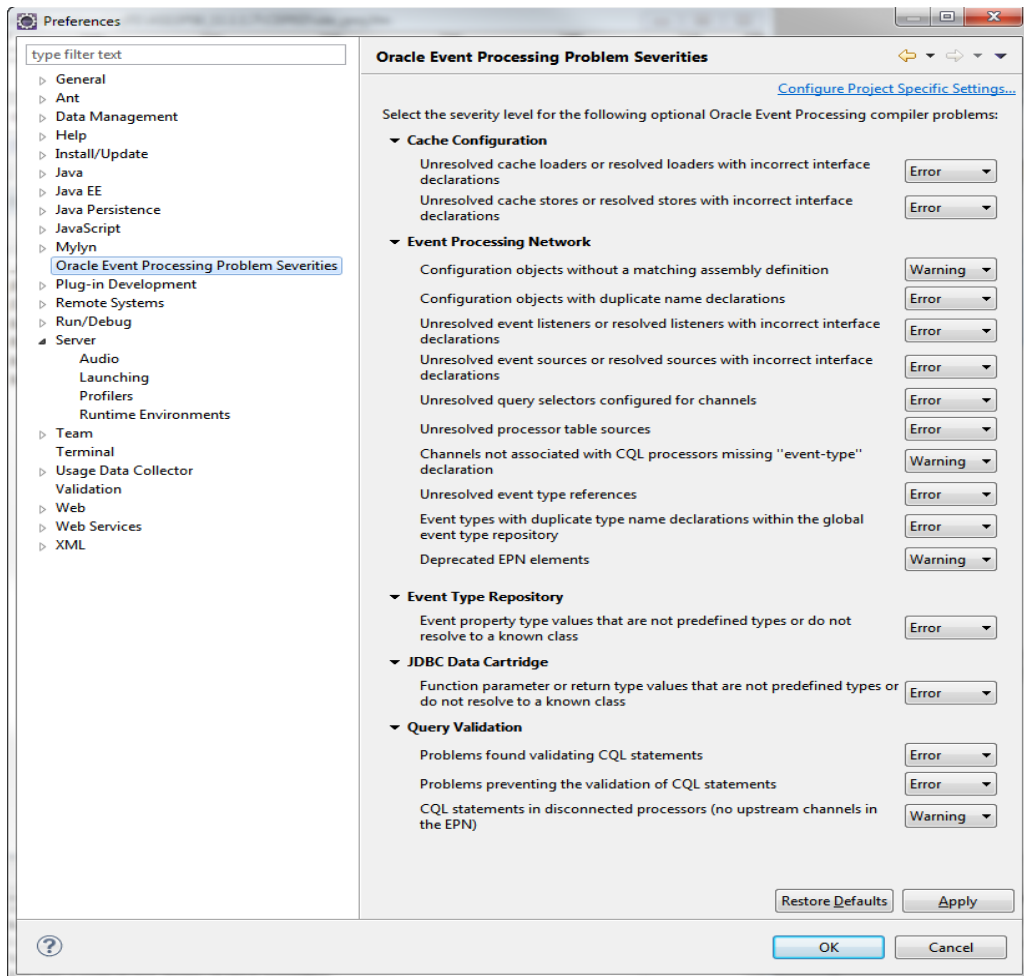
You can configure these preferences for each project individually or you can configure these preferences the same for all the projects in a given workspace.

#### To configure problem severity preferences:

1. Open the EPN Editor (see [Section , "Opening the EPN Editor"](#))
2. Select **Window > Preferences**.  
The Preferences dialog appears.
3. Select **Oracle Event Processing Problem Severities**.

The Oracle Event Processing Problem Severities dialog appears as [Figure 5-42](#) shows.

**Figure 5–42 Oracle Event Processing Problem Severities Dialog: Workspace**



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 5–7 describes each of the problem areas you can configure.

**Table 5–7 Oracle Event Processing Problem Severities**

Category	Problem	Description
Cache Configuration	Unresolved cache loaders or resolved loaders with incorrect interface declarations.	Ensure that the assembly file contains a bean element that identifies the cache loader class for each <code>wlevs:cache-loader</code> element and ensure that the cache loader class implements the appropriate interfaces. For more information, see: <ul style="list-style-type: none"> <li>Section , "Loading Cache Data from a Read-Only Data Source"</li> </ul>
	Unresolved cache stores or resolved stores with incorrect interface declarations.	Ensure that the assembly file contains a bean element that identifies the cache store class for each <code>wlevs:cache-store</code> element and ensure that the cache store class implements the appropriate interfaces. For more information, see: <ul style="list-style-type: none"> <li>Section , "Exchanging Data with a Read-Write Data Source."</li> </ul>
Event Processing Network	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 Event Processing 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.
	Unresolved query selectors configured for channels.	Given a channel with an upstream Oracle CQL processor that defines more than one rule, ensure that the channel component configuration file <code>selector</code> element contains only rule names of the rules defined in the upstream Oracle CQL processor. For more information, see Section , "selector".
	References to foreign stages (stages defined in external applications).	Ensure that references to foreign stages can be resolved. For more information, see Section , "Referencing Foreign Stages in an EPN Assembly File".
	Channels not associated with Oracle CQL processors missing "event-type" declaration.	Given a channel that does not have an upstream Oracle CQL processor, ensure that the assembly file <code>wlevs:channel</code> element is configured with an <code>event-type</code> attribute. For more information, see Chapter 10, "Connecting EPN Stages Using Channels".
	Deprecated EPN elements	Oracle Event Processing provides backwards compatibility with applications built for previous versions. Validate an application's use of deprecated XML elements.
Event Type Repository	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 <code>properties</code> element. The <code>property</code> values are limited to a fixed set of supported types. Validate that the <code>property</code> type is one of these allowed types. For more information, see Chapter 9, "Defining and Using Event Types".
Query Validation	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.

**Table 5–7 (Cont.) Oracle Event Processing Problem Severities**

Category	Problem	Description
	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.
	CQL statements affected directly or indirectly by missing binding parameters.	Parameterized queries.
	CQL statements in disconnected processors (no upstream channels in the EPN).	Ensure that all Oracle CQL processors are connected to an upstream stage on the EPN. Without an upstream stage, the Oracle CQL processor's rules have no event stream to operate on.

5. Click **Apply**.
6. Click **OK**.

---



---

## Oracle Event Processing IDE for Eclipse and Oracle Event Processing Servers

This chapter describes how to use the Oracle Event Processing IDE for Eclipse to create and manage Oracle Event Processing servers to develop and debug event-driven applications.

This chapter includes the following sections:

- [Oracle Event Processing Server Overview](#)
- [Creating Oracle Event Processing Servers](#)
- [Managing Oracle Event Processing Servers](#)
- [Debugging an Oracle Event Processing Application Running on an Oracle Event Processing Server](#)

### Oracle Event Processing Server Overview

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

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

[Table 6–1](#) maps Eclipse terminology used by the Oracle Event Processing IDE for Eclipse to Oracle Event Processing server terminology.

**Table 6–1** *Eclipse and Oracle Event Processing Server Concepts*

Eclipse IDE Concept	Oracle Event Processing Server Concept	Description
Runtime	Oracle Event Processing server installation	The Oracle Event Processing IDE for Eclipse has the concept of a runtime. The runtime defines the location where the Oracle Event Processing IDE for Eclipse can find the installation of a particular Oracle Event Processing 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 6–1 (Cont.) Eclipse and Oracle Event Processing Server Concepts**

Eclipse IDE Concept	Oracle Event Processing Server Concept	Description
Server and Server Instance	Domain	<p>The Oracle Event Processing IDE for Eclipse uses the term Server to describe an actual runnable Oracle Event Processing server instance. You can think of it as something that has start scripts, for example. In Oracle Event Processing server terminology, this equates to a Domain. When you set up a server, you specify the domain that this instance will run.</p> <p>For more information on domains, see:</p> <ul style="list-style-type: none"> <li>■ For more information, see "Administering Oracle Event Processing Standalone-Server Domains" in the <i>Oracle Fusion Middleware Administrator's Guide for Oracle Event Processing</i>.</li> <li>■ For more information, see "Administering Oracle Event Processing Standalone-Server Domains" in the <i>Oracle Fusion Middleware Administrator's Guide for Oracle Event Processing</i>.</li> </ul>
Publish	Deploy	The Oracle Event Processing 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 Event Processing IDE for Eclipse becomes a single Oracle Event Processing application packaged as an OSGi bundle. It is deployed to a server and shows in the Oracle Event Processing server's <code>deployments.xml</code> file.

Server definitions are the central concept in controlling an Oracle Event Processing server from the Oracle Event Processing 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 Event Processing IDE for Eclipse. For more information, see [Section , "Creating Oracle Event Processing Servers"](#).

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

- Start a server from within Oracle Event Processing IDE for Eclipse.
 

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

For more information, see:

  - [Section , "How to Start a Local Oracle Event Processing Server"](#)
  - [Section , "How to Stop a Local Oracle Event Processing Server"](#)
- Attach to a running Oracle Event Processing server.
 

In this case, the user starts the Oracle Event Processing server from the command line, then clicks the **Start** button for that server in Oracle Event Processing IDE for Eclipse. A dialog is shown asking whether or not to attach and, if the user clicks **Yes**, Oracle Event Processing IDE for Eclipse enters attached mode. All of the Oracle Event Processing server features except debug are available. However, the Oracle Event Processing server process is not managed by the Oracle Event

Processing IDE for Eclipse. Clicking the **Stop** button simply disconnects from the attached Oracle Event Processing server; it does not terminate the actual Oracle Event Processing server process. Console messages from the Oracle Event Processing server are sent to the Oracle Event Processing server console (standard output to the terminal window in which it is running). Oracle Event Processing IDE for Eclipse only shows limited Oracle Event Processing IDE for Eclipse operation messages in the console view.

For more information, see:

- [Section , "How to Attach to an Existing Local Oracle Event Processing Server Instance"](#)
- [Section , "How to Detach From an Existing Oracle Event Processing Server Instance"](#)

## Creating Oracle Event Processing Servers

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

You can create a local or remote Oracle Event Processing server:

- **Local server:** a local Oracle Event Processing server is one in which both the server and server runtime are on the same host
- **Remote server:** a remote Oracle Event Processing server is one in which the server and server runtime are on different hosts. The server is on a remote host and the server runtime is on the local host (the host on which you are executing the Oracle Event Processing IDE for Eclipse).

This section describes:

- [Section , "How to Create a Local Oracle Event Processing Server and Server Runtime"](#)
- [Section , "How to Create a Remote Oracle Event Processing Server and Server Runtime"](#)
- [Section , "How to Create an Oracle Event Processing Server Runtime"](#)

### How to Create a Local Oracle Event Processing Server and Server Runtime

This section describes how to create both a local server and server runtime. After creating the initial server and server runtime, you can create additional server runtimes.

A local Oracle Event Processing server is one in which both the server and server runtime are on the same host. Alternatively, you can create a remote server and server runtime.

**Note:** If the server you're creating in Eclipse is one you created from a domain with the Configuration Wizard, be sure to run the server from the command line *before* adding the server to your Eclipse project. Doing so will ensure that all server artifacts are created. Also, when specifying domain configuration information in the Eclipse **New Server** wizard, be sure to click the **Advanced** tab to specify the user name and password used when creating the domain.

For more information on running a server from the command line, see "Starting and Stopping an Oracle Event Processing Server in a Standalone-Server Domain" in *Oracle Fusion Middleware Administrator's Guide for Oracle Event Processing*.

For more information, see:

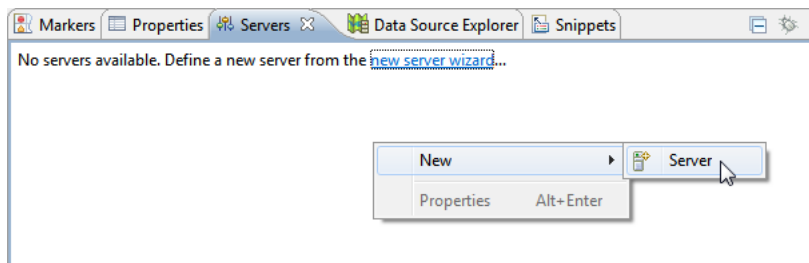
- [Section , "How to Create an Oracle Event Processing Server Runtime"](#)
- [Section , "How to Create a Remote Oracle Event Processing Server and Server Runtime"](#)

**To create a local Oracle Event Processing server and server runtime:**

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

The Servers view appears as shown in [Figure 6–1](#).

**Figure 6–1 Oracle Event Processing IDE for Eclipse Server View**

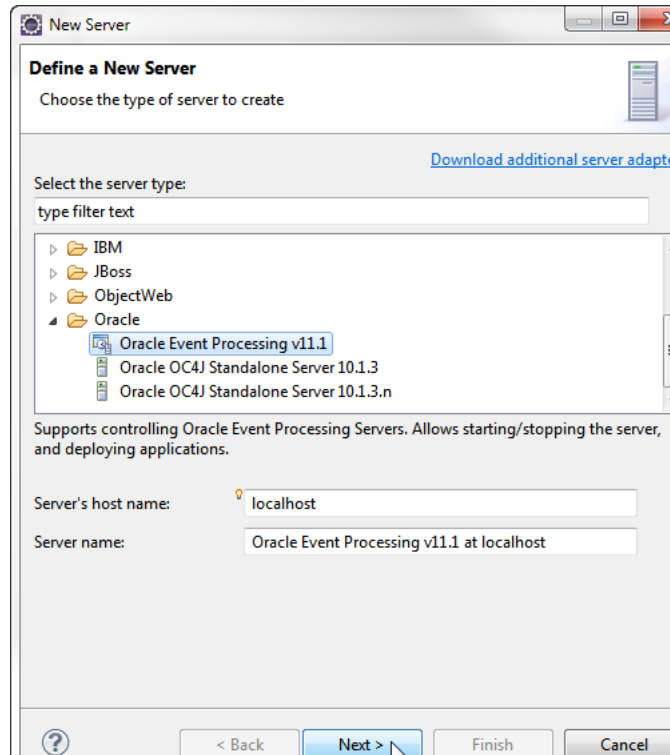


2. Right-click in the **Servers** view pane and select **New > Server**.
3. Consider whether or not server runtimes have been created:
  - a. If this is the first time you have created an Oracle Event Processing server, there will be no installed server runtimes. Proceed to step 4.
  - b. If this is not the first time you have created an Oracle Event Processing server, there will be one or more installed server runtimes. Proceed to step 5.
4. If this is the first time you have created an Oracle Event Processing server, there will be no installed server runtimes:

In this case, the New Server: Define New Server dialog appears as [Figure 6–2](#) shows.



**Figure 6–2 New Server: Define New Server Dialog (No Installed Runtimes)**



Configure the new server as follows:

- a. Configure the dialog as shown in [Table 6–2](#).

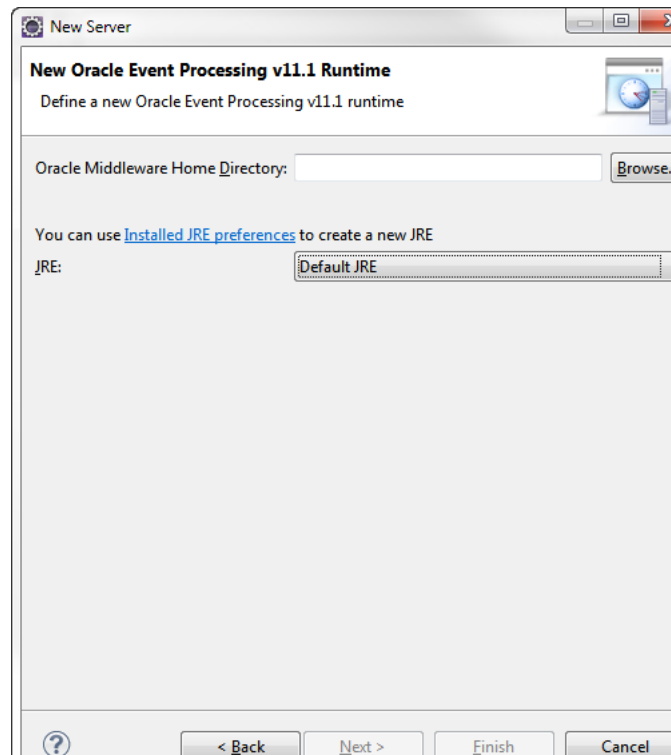
**Table 6–2 New Server: Define New Server Dialog (No Installed Runtimes) Attributes**

Attribute	Description
Server's host name	The host name of the computer on which you installed Oracle Event Processing server. For development, this will typically be localhost.
Select the server type	The type of Oracle Event Processing server. In this example, choose Oracle Event Processing v11
Server name	The name of this Oracle Event Processing server. Default: Oracle Event Processing v11.1 at <i>HOSTNAME</i> Where <i>HOSTNAME</i> is the value you entered in the <b>Server's host name</b> field.

- b. Click **Next**.

The New Server: New Oracle Event Processing v11 Runtime dialog appears as shown in [Figure 6–3](#).

**Figure 6–3 New Server: New Oracle Event Processing v11.1 Runtime Dialog**



- c. Configure the dialog as shown in [Table 6–3](#).

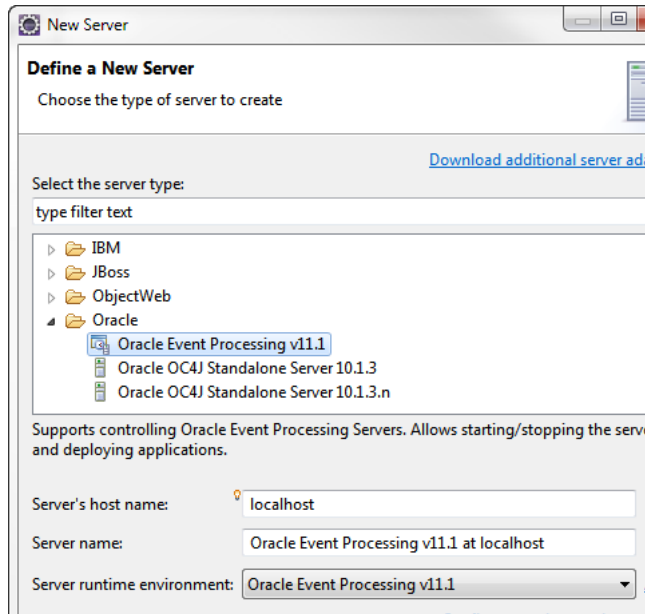
**Table 6–3 New Server: New Oracle Event Processing v11 Runtime Dialog Attributes**

Attribute	Description
Oracle Middleware Home Directory	<p>The fully qualified path to the Oracle Event Processing server installation directory. This is the same as the "Middleware Home" that was selected when installing the server.</p> <p>When selecting this directory, select the directory that contains the Oracle Event Processing installation rather than the Oracle Event Processing directory itself. For example, choose:</p> <p>C:\OracleCEP</p> <p>But do not choose:</p> <p>C:\OracleCEP\ocep_11.1</p> <p>The runtime wizard will use the installation to find the appropriate Oracle Event Processing installation directory.</p> <p>For more information, see "Oracle Fusion Middleware Directory Structure and Concepts" in the <i>Oracle Fusion Middleware Getting Started Guide for Oracle Event Processing</i>.</p>
JRE	<p>The type of JRE to use.</p> <p>Select the type of JRE to use from the pull-down menu or click the <b>Installed JRE preferences</b> link to create a new JRE.</p> <p>Be sure to choose a Java 6 JRE.</p> <p><b>NOTE:</b> The Oracle Event Processing server JRE is ultimately set by the <code>JAVA_HOME</code> setting in <code>setDomainEnv.cmd</code> or <code>setDomainEnv.sh</code> script in the server domain directory.</p>

- d. Proceed to step 6.
- 5. If this is not the first time you have created an Oracle Event Processing server, there will be one or more installed server runtimes.

In this case, the New Server: Define New Server dialog appears as [Figure 6–4](#) shows.

**Figure 6–4 New Server: Define New Server (Installed Runtimes) Dialog**



Configure the dialog as shown in [Table 6–4](#).

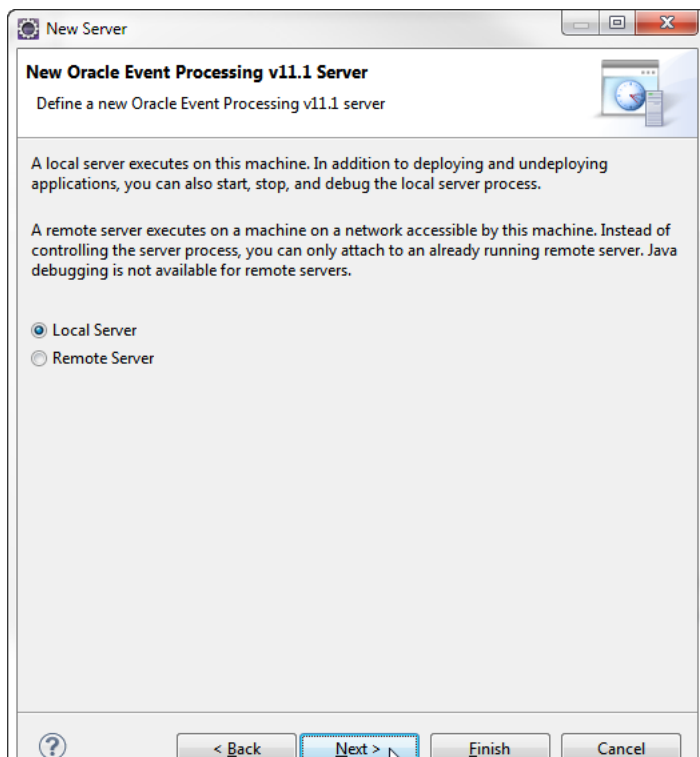
**Table 6–4 New Server: Define New Server (Installed Runtimes) Dialog Attributes**

Attribute	Description
Server's host name	The host name of the computer on which you installed Oracle Event Processing server. For development, this will typically be localhost.
Select the server type	The type of Oracle Event Processing server. In this example, choose Oracle Event Processing v11.
Server runtime	Select the server runtime from the pull-down menu. To create or edit server runtimes, click <b>Installed Runtimes</b> . For more information, see <a href="#">Section , "How to Create an Oracle Event Processing Server Runtime"</a> .

- 6. Click Next.

The New Server: New Oracle Event Processing v11.1 Server dialog appears as [Figure 6–5](#) shows.

**Figure 6–5** *New Server: New Oracle Event Processing v11.1 Server*

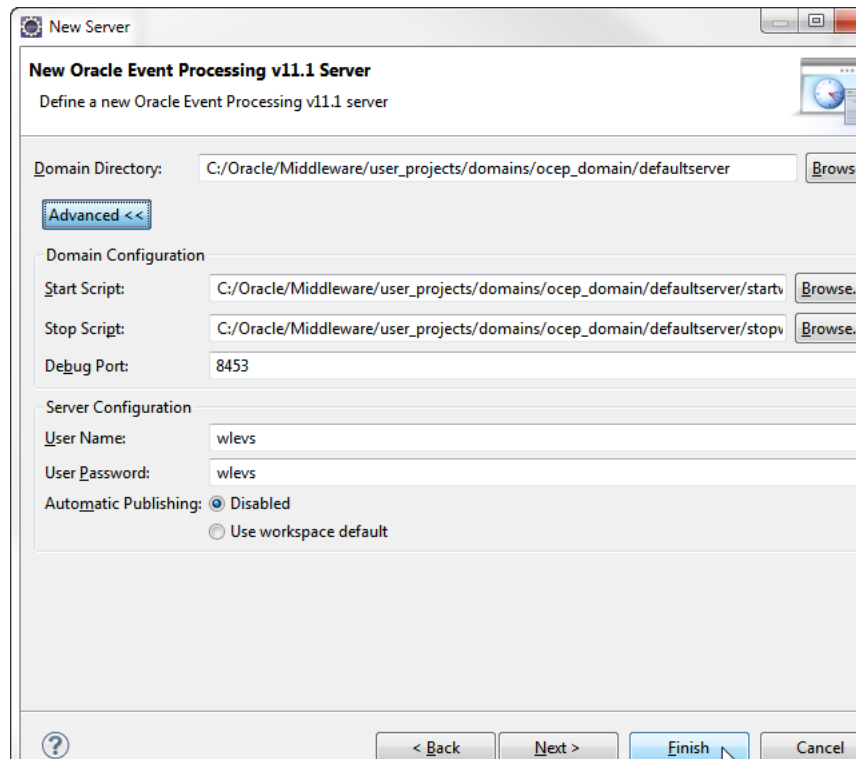


7. Select **Local Server**.

8. Click **Next**.

The New Server: New Oracle Event Processing v11 Server dialog appears as [Figure 6–6](#) shows.

**Figure 6–6 New Server: New Oracle Event Processing v11 Server Dialog for a Local Server**



9. Click **Advanced** and configure the dialog as shown in [Table 6–5](#).

**Table 6–5 New Server: New Oracle Event Processing v11 Server Dialog Attributes for a Local Server**

Attribute	Description
Domain Directory	The fully qualified path to the directory that contains the domain for this server. Click <b>Browse</b> to choose the directory. Default: <code>MIDDLEWARE_HOME\user_projects\domains\ocep_domain\defaultserver</code> .
Start Script	The script that Oracle Event Processing IDE for Eclipse uses to start the Oracle Event Processing server. Default on UNIX: <code>MIDDLEWARE_HOME/user_projects/domains/ocep_domain/defaultserver/startwlevs.sh</code> Default on Windows: <code>MIDDLEWARE_HOME\user_projects\domains\ocep_domain\defaultserver\startwlevs.cmd</code>
Stop Script	The script that Oracle Event Processing IDE for Eclipse uses to stop the Oracle Event Processing server. Default on UNIX: <code>MIDDLEWARE_HOME/user_projects/domains/ocep_domain/defaultserver/stopwlevs.sh</code> Default on Windows: <code>MIDDLEWARE_HOME\user_projects\domains\ocep_domain\defaultserver\stopwlevs.cmd</code>
Debug Port	The Oracle Event Processing server port that Oracle Event Processing IDE for Eclipse connects to when debugging the Oracle Event Processing server. Default: 8453.

**Table 6–5 (Cont.) New Server: New Oracle Event Processing v11 Server Dialog Attributes for a Local Server**

Attribute	Description
User Name	The user name Oracle Event Processing IDE for Eclipse uses when logging into the Oracle Event Processing server. Default: wlevs.
User Password	The user password Oracle Event Processing IDE for Eclipse uses when logging into the Oracle Event Processing server. Default: wlevs.
Automatic Publishing	By default, when you change an application, you must manually publish the changes to the Oracle Event Processing server. Select <b>Use Workspace Default</b> to configure Oracle Event Processing IDE for Eclipse to automatically publish changes to the Oracle Event Processing server. Default: Disabled.

10. Click **Finish**.

11. If you configured **Automatic Publishing** to **Use Workspace Default**, select **Windows > Preferences**.

12. Select the **Server** option.

13. 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 Event Processing 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 Event Processing IDE for Eclipse publishes changes to the server (in seconds).  
Default: 60 seconds.

14. Click **OK**.

## How to Create a Remote Oracle Event Processing Server and Server Runtime

This section describes how to create both a remote server and server runtime. After creating the initial server and server runtime, you can create additional server runtimes.

A remote Oracle Event Processing server is one in which the server and server runtime are on different hosts. The server is on a remote host and the server runtime is on the local host (the host on which you are executing the Oracle Event Processing IDE for Eclipse).

Alternatively, you can create a local server and server runtime.

For more information, see:

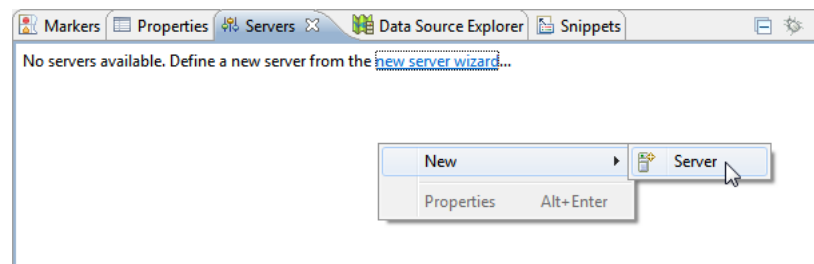
- [Section , "How to Create an Oracle Event Processing Server Runtime"](#)
- [Section , "How to Create a Local Oracle Event Processing Server and Server Runtime"](#)

**To create a remote Oracle Event Processing server and server runtime:**

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

The Servers view appears as shown in [Figure 6–1](#).

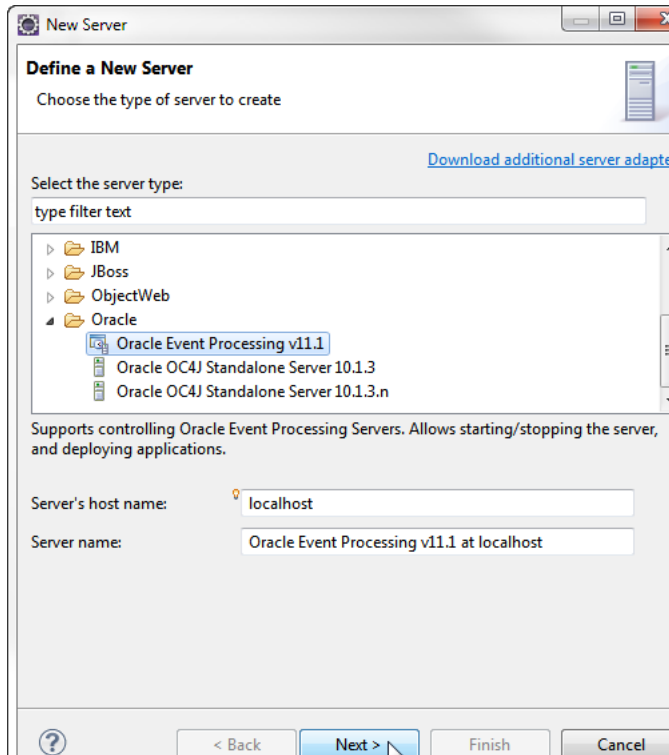
**Figure 6–7 Oracle Event Processing IDE for Eclipse Server View**



2. Right-click in the **Servers** view pane and select **New > Server**.
3. Consider whether or not server runtimes have been created:
  - a. If this is the first time you have created an Oracle Event Processing server, there will be no installed server runtimes. Proceed to step 4.
  - b. If this is not the first time you have created an Oracle Event Processing server, there will be one or more installed server runtimes. Proceed to step 5.
4. If this is the first time you have created an Oracle Event Processing server, there will be no installed server runtimes:

In this case, the New Server: Define New Server dialog appears as [Figure 6–2](#) shows.

**Figure 6–8 New Server: Define New Server Dialog (No Installed Runtimes)**



Configure the new server as follows:

- a. Configure the dialog as shown in [Table 6–2](#).

**Table 6–6 New Server: Define New Server Dialog (No Installed Runtimes) Attributes**

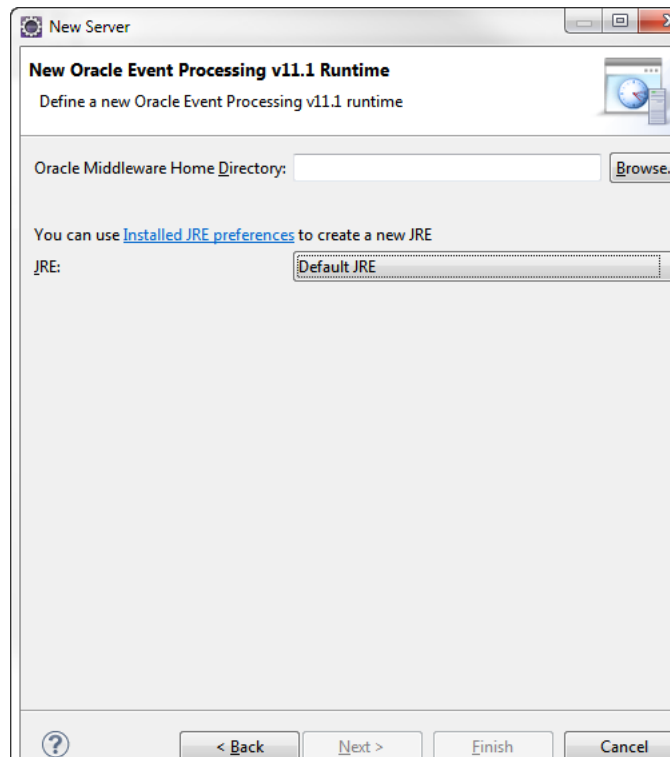
Attribute	Description
Server's host name	The host name of the computer on which you installed Oracle Event Processing server. For development, this will typically be localhost.
Select the server type	The type of Oracle Event Processing server. In this example, choose Oracle Event Processing v11
Server name	The name of this Oracle Event Processing server. Default: Oracle Event Processing v11.1 at <i>HOSTNAME</i> Where <i>HOSTNAME</i> is the value you entered in the <b>Server's host name</b> field.

- b. Click **Next**.

The New Server: New Oracle Event Processing v11 Runtime dialog appears as shown in [Figure 6–3](#).



**Figure 6–9 New Server: New Oracle Event Processing v11.1 Runtime Dialog**



- c. Configure the dialog as shown in [Table 6–3](#).

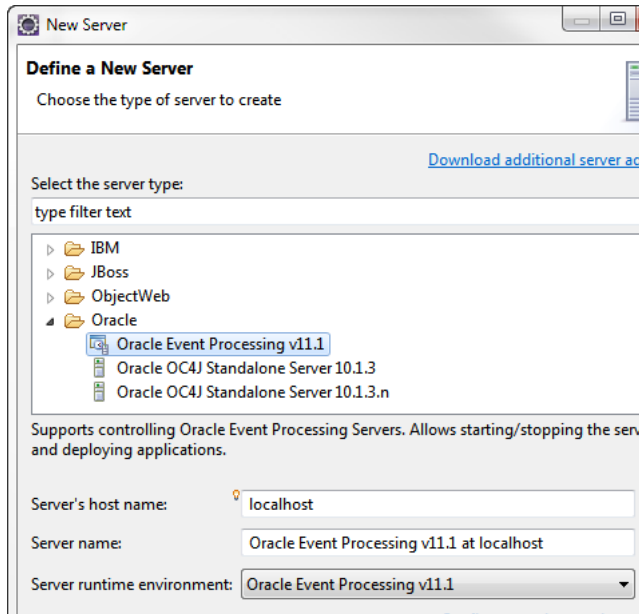
**Table 6–7 New Server: New Oracle Event Processing v11 Runtime Dialog Attributes**

Attribute	Description
Oracle Middleware Home Directory	<p>The fully qualified path to the Oracle Event Processing server installation directory. This is the same as the "Middleware Home" that was selected when installing the server.</p> <p>When selecting this directory, select the directory that contains the Oracle Event Processing installation rather than the Oracle Event Processing directory itself. For example, choose:</p> <p>C:\OracleCEP</p> <p>But do not choose:</p> <p>C:\OracleCEP\ocep_11.1</p> <p>The runtime wizard will use the installation to find the appropriate Oracle Event Processing installation directory.</p> <p>For more information, see "Oracle Fusion Middleware Directory Structure and Concepts" in the <i>Oracle Fusion Middleware Getting Started Guide for Oracle Event Processing</i>.</p>
JRE	<p>The type of JRE to use.</p> <p>Select the type of JRE to use from the pull-down menu or click the <b>Installed JRE preferences</b> link to create a new JRE.</p> <p>Be sure to choose a Java 6 JRE.</p> <p><b>NOTE:</b> The Oracle Event Processing server JRE is ultimately set by the JAVA_HOME setting in setDomainEnv.cmd or setDomainEnv.sh script in the server domain directory.</p>

- d. Proceed to step 6.
- 5. If this is not the first time you have created an Oracle Event Processing server, there will be one or more installed server runtimes.

In this case, the New Server: Define New Server dialog appears as [Figure 6–4](#) shows.

**Figure 6–10 New Server: Define New Server (Installed Runtimes) Dialog**



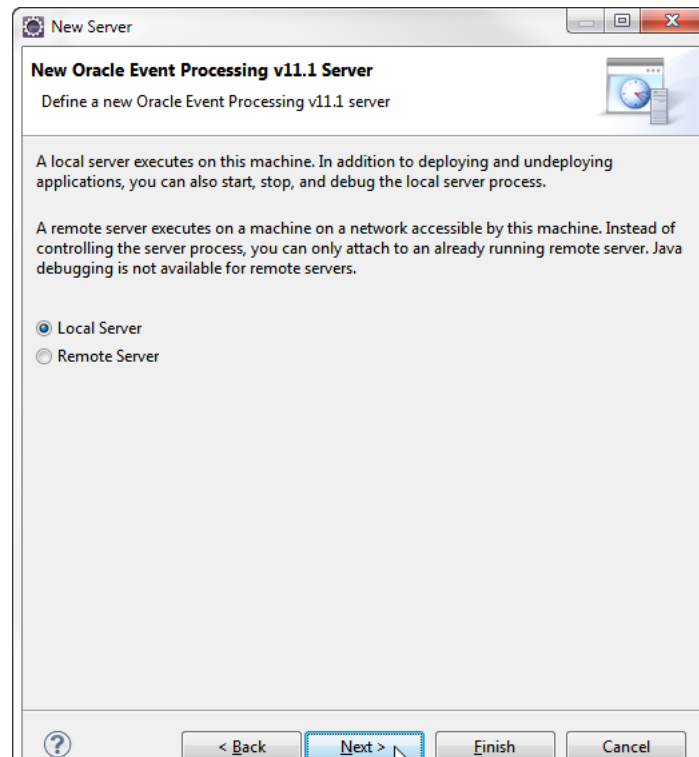
Configure the new server as follows:

**Table 6–8 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 Event Processing server. For development, this will typically be localhost.
Select the server type	The type of Oracle Event Processing server. In this example, choose Oracle Event Processing v11.
Server runtime	Select the server runtime from the pull-down menu. To create or edit server runtimes, click <b>Installed Runtimes</b> . For more information, see <a href="#">Section , "How to Create an Oracle Event Processing Server Runtime"</a> .

- 6. Click **Next**.

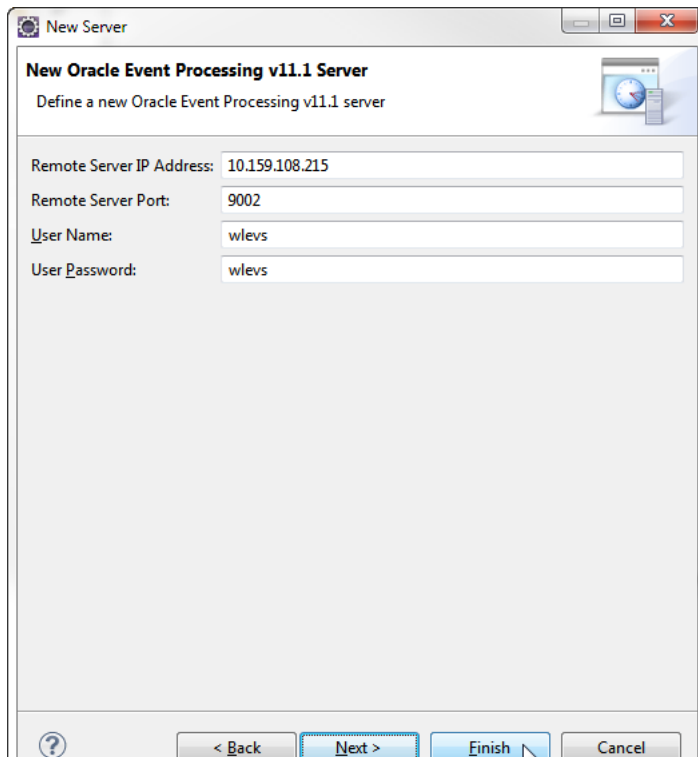
The New Server: New Oracle Event Processing v11.1 Server dialog appears as [Figure 6–5](#) shows.

**Figure 6–11 New Server: New Oracle Event Processing v11.1 Server**

7. Select **Remote Server**.
8. Click **Next**.

The New Server: New Oracle Event Processing v11 Server dialog appears dialog appears as [Figure 6–12](#) shows.

**Figure 6–12 New Server: New Oracle Event Processing v11 Server Dialog for a Remote Server**



9. Configure the dialog as shown in [Table 6–9](#).

**Table 6–9 New Server: Oracle Event Processing v11 Server Dialog Attributes for a Local Server**

Attribute	Description
Remote Server IP Address	The IP address of the remote Oracle Event Processing server. Default: IP address of localhost.
Remote Server Port	The port you specified in the remote Oracle Event Processing server <code>DOMAIN_DIR/config/config.xml</code> file that describes your Oracle Event Processing domain, where <code>DOMAIN_DIR</code> refers to your domain directory.  The port number is the value of the <code>Port</code> child element of the <code>Netio</code> element:  <pre>&lt;Netio&gt;   &lt;Name&gt;NetIO&lt;/Name&gt;   &lt;Port&gt;9002&lt;/Port&gt; &lt;/Netio&gt;</pre> Default: 9002
User Name	The user name that the Oracle Event Processing IDE for Eclipse uses to log into the remote server.  Default: wlevs
User Password	The password that the Oracle Event Processing IDE for Eclipse uses to log into the remote server.  Default: wlevs

10. Click **Finish**.

## How to Create an Oracle Event Processing Server Runtime

Before you can create a server, you must configure the Oracle Event Processing IDE for Eclipse with the location of your Oracle Event Processing 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 Event Processing server.

For more information, see:

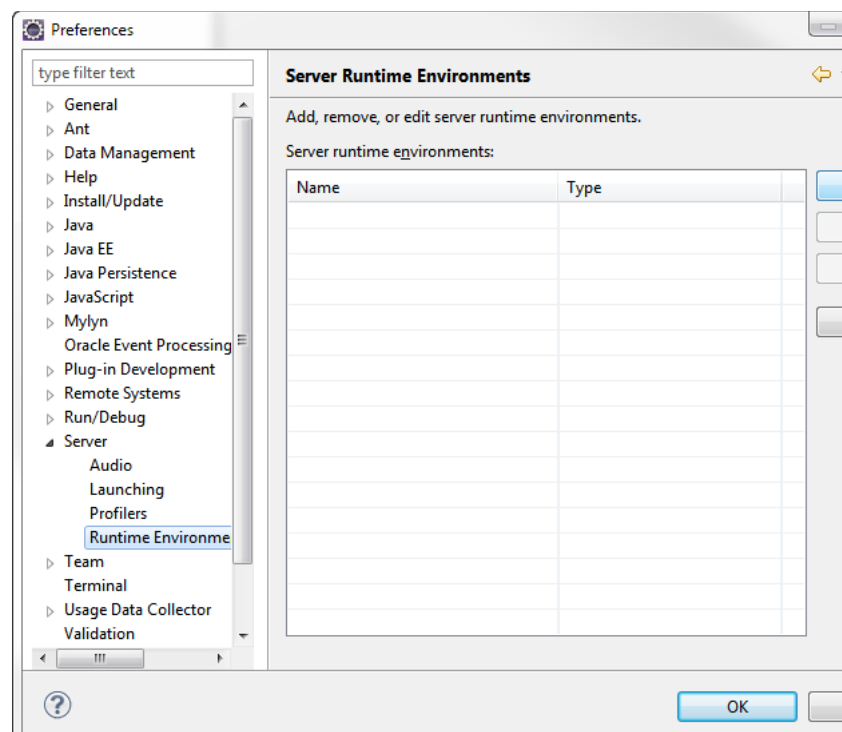
- [Section , "How to Create a Local Oracle Event Processing Server and Server Runtime"](#)
- [Section , "How to Create a Remote Oracle Event Processing Server and Server Runtime"](#)

### To create an Oracle Event Processing server runtime:

1. Select **Windows > Preferences**.

The Preferences dialog appears as [Figure 6–13](#) shows.

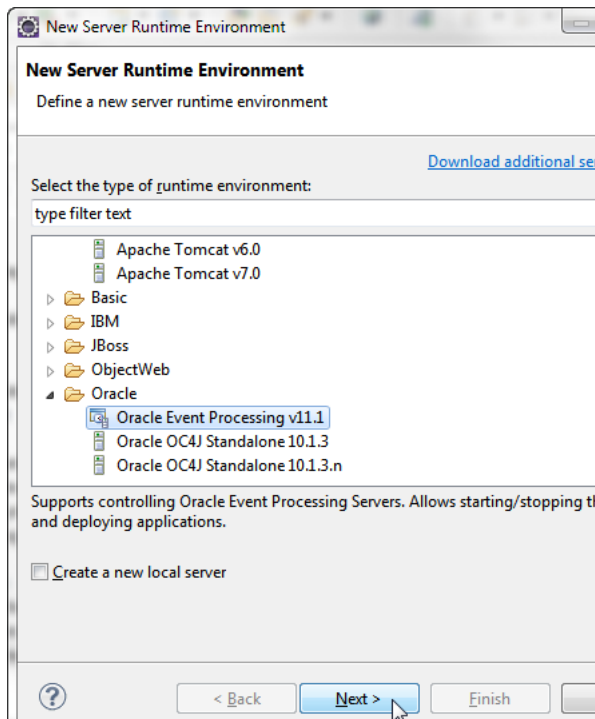
**Figure 6–13 Preferences - Server - Installed Runtimes**



2. Expand the **Server** option and select **Runtime Environments**.
3. Click **Add**.

The New Server Runtime Environment dialog appears as shown in [Figure 6–14](#).

**Figure 6–14 New Server Runtime Environment Dialog**



4. Configure the dialog as shown in [Table 6–10](#).

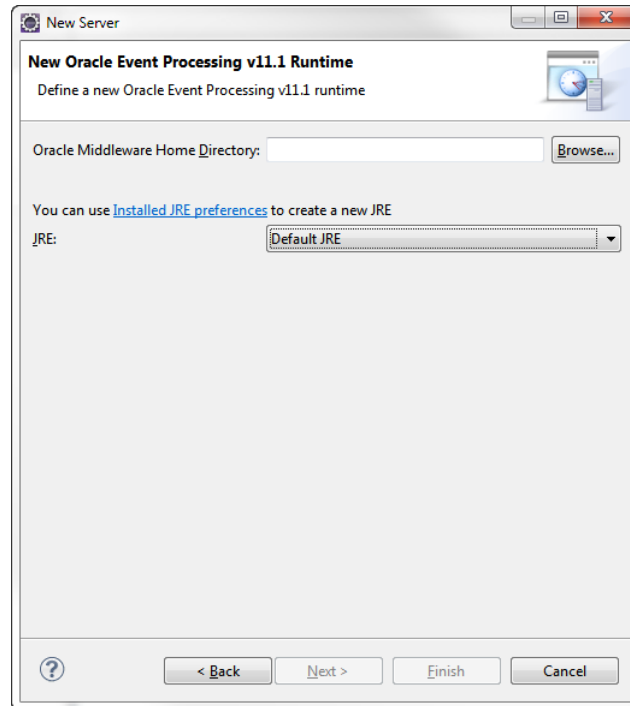
**Table 6–10 New Server Runtime Dialog Attributes**

Attribute	Description
Select the type of runtime environment	The type of Oracle Event Processing server. In this example, choose Oracle Event Processing v11.1.
Create a new local server	Optionally, check this to create a new local server if you have not yet created a server. For more information, see <a href="#">Section , "How to Create a Local Oracle Event Processing Server and Server Runtime"</a> .

5. Click **Next**.

The New Server Runtime Environment dialog appears as shown in [Figure 6–15](#).

**Figure 6–15 New Server Runtime Environment: New Oracle Event Processing v11.1 Runtime Dialog**



6. Configure the dialog as shown in [Table 6–11](#).

**Table 6–11 New Server Runtime Dialog Attributes**

Attribute	Description
Oracle Middleware Home Directory	<p>The fully qualified path to the Oracle Event Processing server installation directory. This is the same as the "Middleware Home" that was selected when installing the server.</p> <p>When selecting this directory, select the directory that contains the Oracle Event Processing installation rather than the Oracle Event Processing directory itself. For example, choose:</p> <p>C:\OracleCEP</p> <p>But do not choose:</p> <p>C:\OracleCEP\ocep_11.1</p> <p>The runtime wizard will use the installation to find the appropriate Oracle Event Processing installation directory.</p> <p>For more information, see "Oracle Fusion Middleware Directory Structure and Concepts" in the <i>Oracle Fusion Middleware Getting Started Guide for Oracle Event Processing</i>.</p>
JRE	<p>The type of JRE to use.</p> <p>Select the type of JRE to use from the pull-down menu or click the <b>Installed JRE preferences</b> link to create a new JRE.</p> <p>Be sure to choose either JRockit Real Time or the JRockit JDK installed with your Oracle Event Processing installation.</p>

7. Click **Finish**.

## Managing Oracle Event Processing Servers

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

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

- [Section , "How to Start a Local Oracle Event Processing Server"](#)
- [Section , "How to Stop a Local Oracle Event Processing Server"](#)
- [Section , "How to Attach to an Existing Local Oracle Event Processing Server Instance"](#)
- [Section , "How to Attach to an Existing Remote Oracle Event Processing Server Instance"](#)
- [Section , "How to Detach From an Existing Oracle Event Processing Server Instance"](#)
- [Section , "How to Deploy an Application to an Oracle Event Processing Server"](#)
- [Section , "How to Configure Connection and Control Settings for Oracle Event Processing Server"](#)
- [Section , "How to Configure Domain \(Runtime\) Settings for Oracle Event Processing Server"](#)
- [Section , "How to Start the Oracle Event Processing Visualizer from Oracle Event Processing IDE for Eclipse"](#)

### How to Start a Local Oracle Event Processing Server

After you create a local server, you can start the Oracle Event Processing server from the Oracle Event Processing IDE for Eclipse.

You can also start the local Oracle Event Processing server in debug mode.

Alternatively, you can start the local Oracle Event Processing server from the command line and attach to it using Oracle Event Processing IDE for Eclipse.

For more information, see:

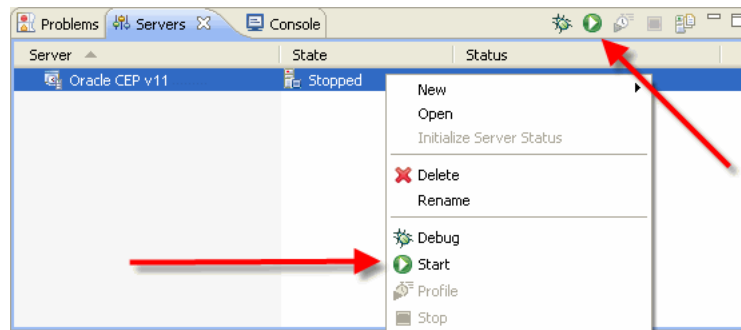
- [Section , "How to Create a Local Oracle Event Processing Server and Server Runtime"](#)
- [Section , "How to Debug an Oracle Event Processing Application Running on an Oracle Event Processing Server"](#)
- [Section , "How to Attach to an Existing Local Oracle Event Processing Server Instance"](#)

#### **To start a local Oracle Event Processing server:**

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

The Servers view opens as shown in [Figure 6–16](#).



**Figure 6–16 Starting an Oracle Event Processing 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.

## How to Stop a Local Oracle Event Processing Server

After you start a local Oracle Event Processing server from the Oracle Event Processing IDE for Eclipse, you can stop the Oracle Event Processing server from the Oracle Event Processing IDE for Eclipse.

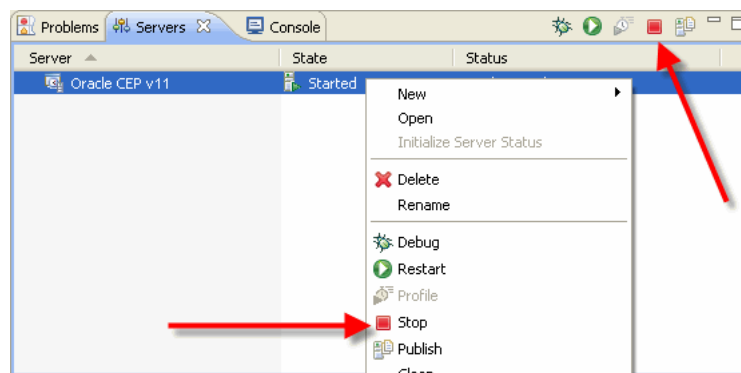
Note that stopping the Oracle Event Processing while Oracle Event Processing Visualizer is running might produce console messages indicating that a service proxy has been destroyed. This is generally an informational message only.

For more information, see [Section , "How to Start a Local Oracle Event Processing Server"](#).

### To stop a local Oracle Event Processing server:

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

The Servers view opens as shown in [Figure 6–17](#).

**Figure 6–17 Stopping an Oracle Event Processing 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**.

## How to Attach to an Existing Local Oracle Event Processing Server Instance

After you create a local server, you can start the local Oracle Event Processing server from the command line and attach Oracle Event Processing IDE for Eclipse to this existing, already running local Oracle Event Processing server instance.

Alternatively, you can start the local Oracle Event Processing server directly from within Oracle Event Processing IDE for Eclipse.

For more information, see:

- [Section , "How to Create a Local Oracle Event Processing Server and Server Runtime"](#)
- [Section , "How to Start a Local Oracle Event Processing Server"](#)

### To attach to an existing local Oracle Event Processing server instance:

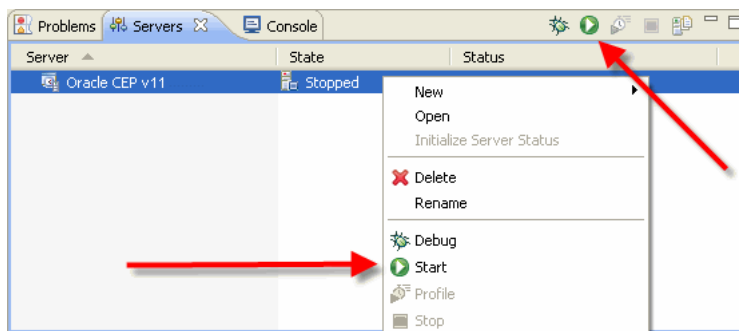
1. Start the Oracle Event Processing server from the command line.

For more information, see "Starting and Stopping Oracle Event Processing Servers" in the *Oracle Fusion Middleware Administrator's Guide for Oracle Event Processing*.

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

The Servers view opens as shown in [Figure 6–16](#).

**Figure 6–18 Attaching to an Existing Local Oracle Event Processing Server Instance**



3. Attach to the already running local 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 OEP Server dialog appears.

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 Event Processing Visualizer. For more information, see "How to View Console Output" in the *Oracle Fusion Middleware Visualizer User's Guide for Oracle Event Processing*.

## How to Attach to an Existing Remote Oracle Event Processing Server Instance

After you create a remote server, you can start the remote Oracle Event Processing server from the command line and attach Oracle Event Processing IDE for Eclipse to this existing, already running remote Oracle Event Processing server instance.

For more information, see [Section , "How to Create a Remote Oracle Event Processing Server and Server Runtime"](#).

**To attach to an existing remote Oracle Event Processing server instance:**

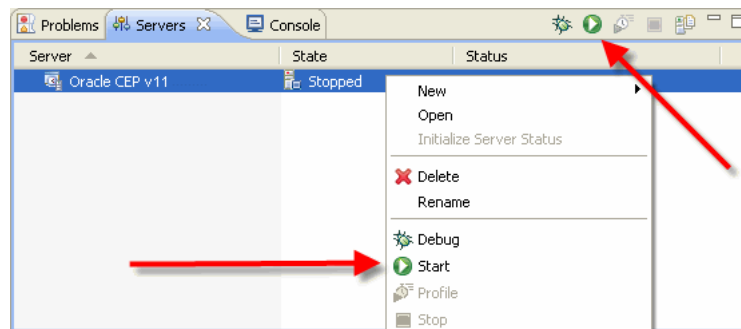
1. Start the remote Oracle Event Processing server from the command line.

For more information, see "Starting and Stopping Oracle Event Processing Servers" in the *Oracle Fusion Middleware Administrator's Guide for Oracle Event Processing*.

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

The Servers view opens as shown in [Figure 6–16](#).

**Figure 6–19 Attaching to an Existing Remote Oracle Event Processing Server Instance**



3. Attach to the already running remote 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 attaching to the remote server, Oracle Event Processing IDE for Eclipse writes one status message to the Console view, reading:

```
[3/23/10 12:32 PM] Attached to remote OEP server at address 10.11.12.13 and port 9002
```

You will *not* see log messages from the remote server in the Console view.

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

## How to Detach From an Existing Oracle Event Processing Server Instance

After you attach to an existing, running Oracle Event Processing server instance, you can detach from the Oracle Event Processing server and leave it running.

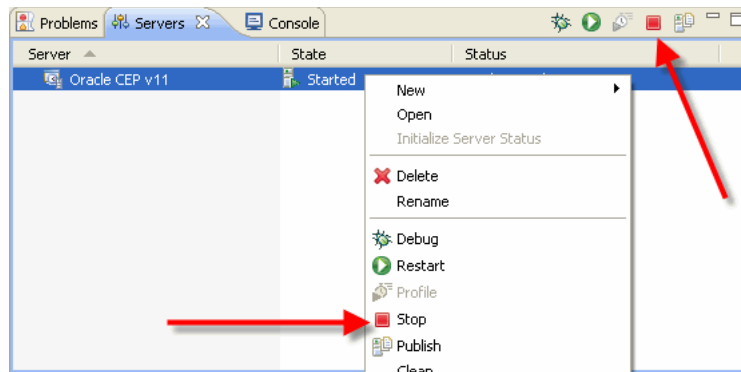
For more information, see:

- [Section , "How to Attach to an Existing Local Oracle Event Processing Server Instance"](#)
- [Section , "How to Attach to an Existing Remote Oracle Event Processing Server Instance"](#)

**To detach from an existing Oracle Event Processing server instance:**

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

The Servers view opens as shown in [Figure 6–17](#).

**Figure 6–20 Stopping an Oracle Event Processing 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 Event Processing IDE for Eclipse detaches from the Oracle Event Processing server instance. The Oracle Event Processing server instance continues to run.

If you detach from a remote Oracle Event Processing server, Oracle Event Processing IDE for Eclipse writes a log message to the Console view reading:

```
[3/23/10 12:47 PM] Server communication stopped
```

## How to Deploy an Application to an Oracle Event Processing Server

A project in the Oracle Event Processing IDE for Eclipse is built as an Oracle Event Processing application, then deployed to the server. To deploy an application, a server must first be defined. 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.

For more information, see:

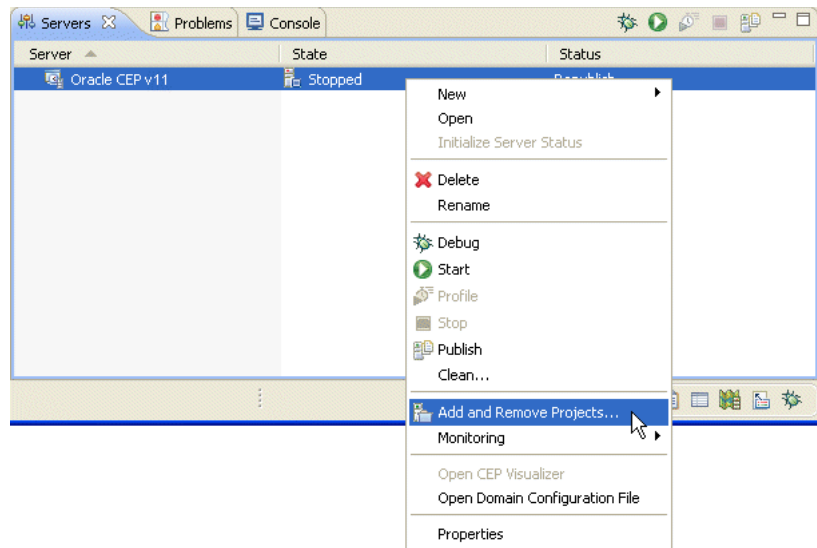
- [Section , "How to Create a Local Oracle Event Processing Server and Server Runtime"](#)
- [Section , "How to Create a Remote Oracle Event Processing Server and Server Runtime"](#)

### To deploy an application to an Oracle Event Processing server:

1. Create an Oracle Event Processing project (see [Section , "Creating Oracle Event Processing Projects"](#)).
2. Create a server (see [Section , "How to Create a Local Oracle Event Processing Server and Server Runtime"](#)).
3. Select **Window > Show Views > Servers**.

The Servers view opens as shown in [Figure 6–21](#).

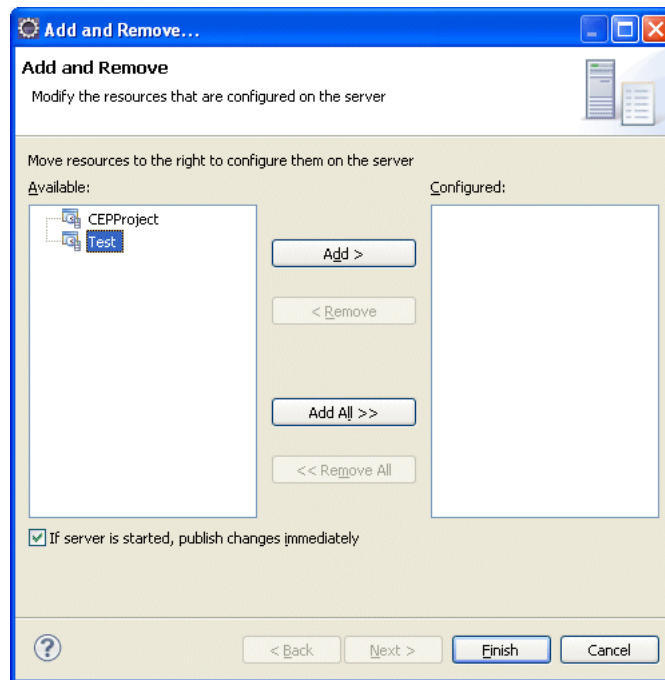
**Figure 6–21 Adding a Project to an Oracle Event Processing Server**



4. Right-click the server and select **Add and Remove**.

The Add and Remove dialog appears as shown in [Figure 6–22](#).

**Figure 6–22 Add and Remove Dialog**



5. Configure the dialog as [Table 6–12](#) shows.

**Table 6–12 Add and Remove Dialog Attributes**

Attribute	Description
Available	Select one or more projects from this list and click <b>Add</b> or <b>Add All</b> to move them into the Configured list.

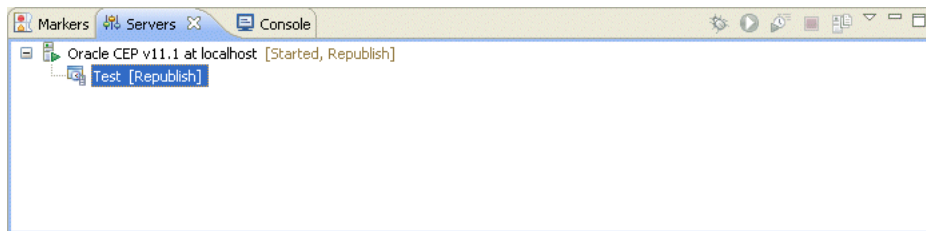
**Table 6–12 (Cont.) Add and Remove Dialog Attributes**

Attribute	Description
Configured	Select one or more projects from this list and click <b>Remove</b> or <b>Remove All</b> to move them into the Available list.
If server is started, publish changes immediately.	Check this option to immediately publish projects that you modify. Applicable only if the server is already running.

6. Click **Finish**.

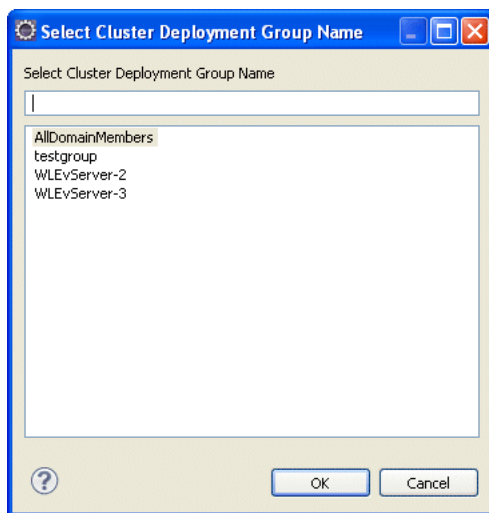
Once an application is added, it will show as a child of the server in the Servers view as shown in [Figure 6–23](#).

**Figure 6–23 Server View After Adding a Project**



7. To deploy (publish) the application to the Oracle Event Processing server, right-click the added application and select **Force Publish**.
  - a. If the Oracle Event Processing server is part of a standalone-server, domain the application is deployed.
  - b. If the Oracle Event Processing server is part of a multi-server domain, the Select Cluster Deployment Group Name dialog appears as [Figure 6–24](#) shows.

**Figure 6–24 Select Cluster Deployment Group Name Dialog**

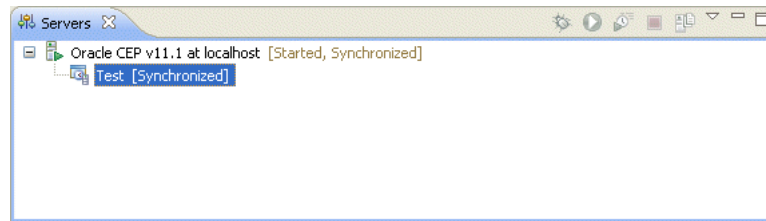


Select the cluster deployment group you want to deploy the application to and click **OK**.

For more information on clustering, see "Introduction to Multi-Server Domains" in the *Oracle Fusion Middleware Administrator's Guide for Oracle Event Processing*.

Once an application is deployed (published), it will show as a child of the server in the Servers view as shown in [Figure 6–25](#).

**Figure 6–25 Server View After Deploying (Publishing) a Project**



## How to Configure Connection and Control Settings for Oracle Event Processing Server

After you create a server, you can use the Server Overview editor to configure all the important server connection and control settings that Oracle Event Processing IDE for Eclipse uses to communicate with the Oracle Event Processing server.

For more information, see:

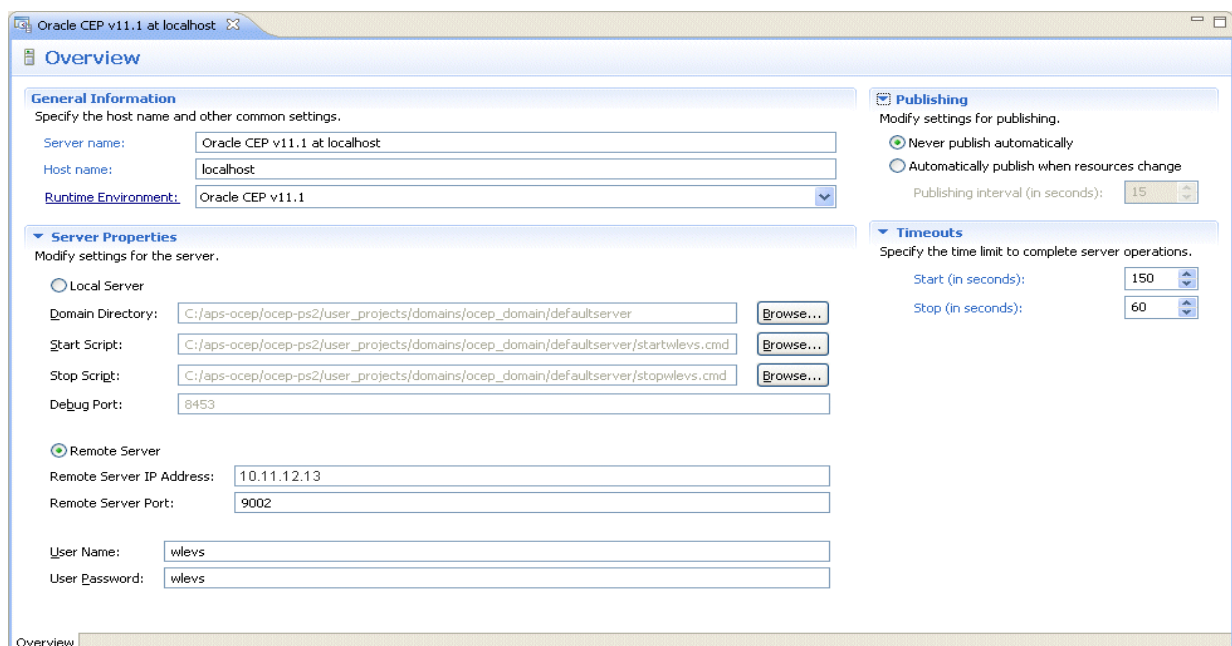
- [Section , "How to Create a Local Oracle Event Processing Server and Server Runtime"](#)
- [Section , "How to Configure Domain \(Runtime\) Settings for Oracle Event Processing Server"](#)

**To configure connection and control settings for Oracle Event Processing 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 6–26](#).

**Figure 6–26 Server Overview Editor**



- Configure the Server Overview editor as shown in [Table 6–13](#).

**Table 6–13 Server Overview Editor Attributes**

Attribute	Description
Server Name	<p>The name of this server. Only used within the Oracle Event Processing IDE for Eclipse as a useful identifier.</p> <p>For more information, see <a href="#">Section , "How to Create a Local Oracle Event Processing Server and Server Runtime"</a>.</p>
Host Name	<p>The name of the host on which this server is installed.</p> <p>For more information, see <a href="#">Section , "How to Create a Local Oracle Event Processing Server and Server Runtime"</a>.</p>
Runtime Environment	<p>The current installed runtime selected for this server.</p> <p>Select a new runtime from the pull down menu or click the <b>Edit</b> link to modify the configuration of the selected runtime.</p> <p>For more information, see <a href="#">Section , "How to Create a Local Oracle Event Processing Server and Server Runtime"</a>.</p>
Domain Directory <sup>1</sup>	<p>The fully qualified path to the directory that contains the domain for this server.</p> <p>Click <b>Browse</b> to choose the directory.</p> <p>Default: <code>MIDDLEWARE_HOME\user_projects\domains\ocep_domain\defaultserver</code>.</p>
Start Script <sup>1</sup>	<p>The script that Oracle Event Processing IDE for Eclipse uses to start the Oracle Event Processing server.</p> <p>Click <b>Browse</b> to choose the start script.</p> <p>Default on UNIX: <code>MIDDLEWARE_HOME\user_projects\domains\ocep_domain\defaultserver\stopwlevs.sh</code></p> <p>Default on Windows: <code>MIDDLEWARE_HOME\user_projects\domains\ocep_domain\defaultserver\stopwlevs.cmd</code></p>
Stop Script <sup>1</sup>	<p>The script that Oracle Event Processing IDE for Eclipse uses to stop the Oracle Event Processing server.</p> <p>Click <b>Browse</b> to choose the stop script.</p> <p>Default on UNIX: <code>MIDDLEWARE_HOME\user_projects\domains\ocep_domain\defaultserver\startwlevs.sh</code></p> <p>Default on Windows: <code>MIDDLEWARE_HOME\user_projects\domains\ocep_domain\defaultserver\startwlevs.cmd</code></p>
Debug Port <sup>1</sup>	<p>The Oracle Event Processing server port that Oracle Event Processing IDE for Eclipse connects to when debugging the Oracle Event Processing server.</p> <p>Default: 8453.</p>
Remote Server IP Address <sup>2</sup>	<p>The IP address of the remote Oracle Event Processing server.</p> <p>Default: IP address of localhost.</p>



**Table 6–13 (Cont.) Server Overview Editor Attributes**

Attribute	Description
Remote Server Port <sup>2</sup>	<p>The port you specified in the remote Oracle Event Processing server <code>DOMAIN_DIR/config/config.xml</code> file that describes your Oracle Event Processing domain, where <code>DOMAIN_DIR</code> refers to your domain directory.</p> <p>The port number is the value of the <code>Port</code> child element of the <code>Netio</code> element:</p> <pre>&lt;Netio&gt;   &lt;Name&gt;NetIO&lt;/Name&gt;   &lt;Port&gt;9002&lt;/Port&gt; &lt;/Netio&gt;</pre> <p>Default: 9002</p>
User Name <sup>2</sup>	<p>The user name that the Oracle Event Processing IDE for Eclipse uses to log into the remote server.</p> <p>Default: <code>wlevs</code></p>
User Password <sup>2</sup>	<p>The password that the Oracle Event Processing IDE for Eclipse uses to log into the remote server.</p> <p>Default: <code>wlevs</code></p>
Publishing	<p>By default, when you change an application, you must manually publish the changes to the Oracle Event Processing server.</p> <p>Select <b>Never publish automatically</b> to disable automatic publishing.</p> <p>Select <b>Override default settings</b> to override the default automatic publishing interval. Enter a new publishing interval (in seconds).</p> <p>Default: Never publish automatically.</p>
Timeouts	<p>Enter a positive, integer number of seconds in the <b>Start (in seconds)</b> field to specify the time in which the Oracle Event Processing server must start.</p> <p>Default: 150 seconds.</p> <p>Enter a positive, integer number of seconds in the <b>Stop (in seconds)</b> field to specify the time in which the Oracle Event Processing server must start.</p> <p>Default: 60 seconds.</p>

<sup>1</sup> Click **Local Server** to modify. Applies to both a local server and the runtime of a remote server.

<sup>2</sup> Click **Remote Server** to modify. Applies only to a remote server.

4. Select **File > Save**.
5. Close the Server Overview editor.

## How to Configure Domain (Runtime) Settings for Oracle Event Processing Server

After you create a server, you can use the Oracle Event Processing IDE for Eclipse to configure Oracle Event Processing server domain (runtime) settings in the Oracle Event Processing server `config.xml` file.

Recall that a local Oracle Event Processing server is one in which both the server and server runtime are on the same host and a remote Oracle Event Processing server is one in which the server and server runtime are on different hosts: the server is on a

remote host and the server runtime is on the local host (the host on which you are executing the Oracle Event Processing IDE for Eclipse).

For both local and remote Oracle Event Processing servers, when you configure domain (runtime) settings, you are modifying only the Oracle Event Processing server `config.xml` on the local host.

You can also use the Oracle Event Processing IDE for Eclipse to configure all the important server connection and control settings that Oracle Event Processing IDE for Eclipse uses to communicate with the Oracle Event Processing server.

Any changes you make to the Oracle Event Processing server `config.xml` file for a running Oracle Event Processing server are not read by the Oracle Event Processing server until you restart it.

If you make changes to the Oracle Event Processing server `config.xml` file for a running Oracle Event Processing server using the Oracle Event Processing Visualizer, the changes apply to the running Oracle Event Processing server as soon as you save them. The Oracle Event Processing Visualizer updates the Oracle Event Processing server `config.xml` file and overwrites the current filesystem version of that file with the current, in-memory version.

If you make changes to the Oracle Event Processing server `config.xml` file by manually editing this file, and you then make further changes using the Oracle Event Processing Visualizer, your manual edits will be overwritten by the Oracle Event Processing Visualizer.

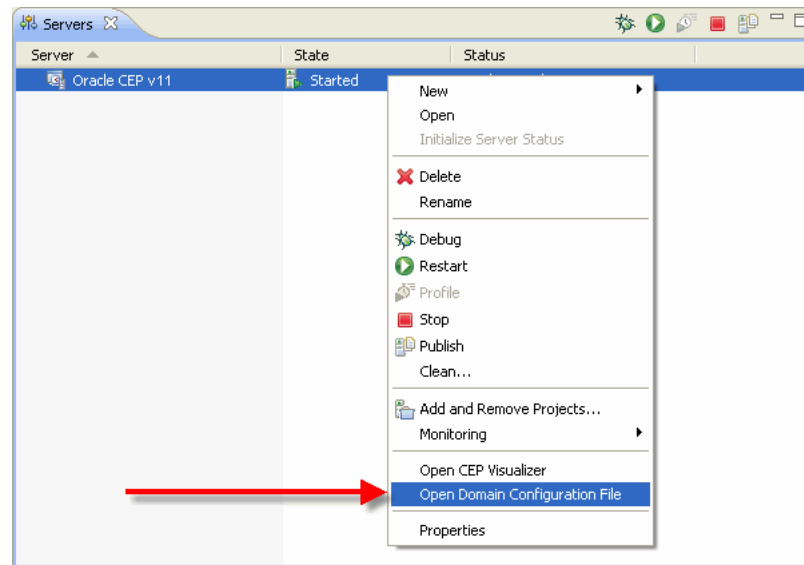
To avoid this, when you manually edit the Oracle Event Processing server `config.xml` file, always stop and start the Oracle Event Processing server to read those changes into the runtime configuration and then use the Oracle Event Processing Visualizer to make further changes.

For more information, see:

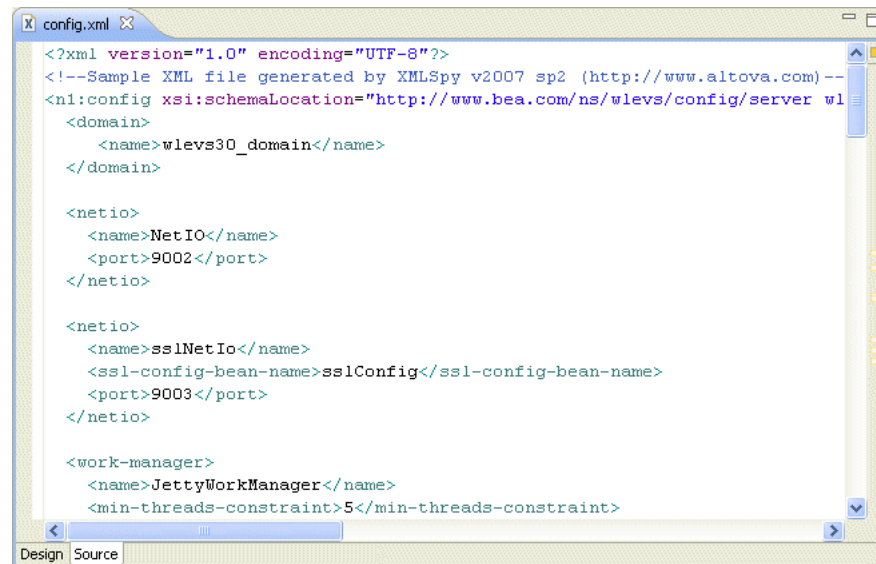
- [Section , "How to Create a Local Oracle Event Processing Server and Server Runtime"](#)
- [Section , "How to Create a Remote Oracle Event Processing Server and Server Runtime"](#)
- [Section , "How to Configure Connection and Control Settings for Oracle Event Processing Server"](#)
- [Section , "How to Start the Oracle Event Processing Visualizer from Oracle Event Processing IDE for Eclipse"](#)
- "Understanding Oracle Event Processing Server Configuration" in the *Oracle Fusion Middleware Administrator's Guide for Oracle Event Processing*

**To configure domain (runtime) settings for Oracle Event Processing 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 6-27](#).

**Figure 6–27 Editing the Domain Configuration File**

The Oracle Event Processing server domain configuration file `config.xml` opens as shown in [Figure 6–28](#).

**Figure 6–28 Oracle Event Processing Domain Configuration File `config.xml`**

3. Edit the domain configuration file as required.
4. Select **File > Save**.
5. Close the domain configuration file.

## How to Start the Oracle Event Processing Visualizer from Oracle Event Processing IDE for Eclipse

After you create a server, you can start the Oracle Event Processing Visualizer from the Oracle Event Processing IDE for Eclipse.

The Oracle Event Processing Visualizer is the administration console for a running Oracle Event Processing server. For more information, see the Oracle Fusion Middleware Visualizer User's Guide for Oracle Event Processing.

For more information, see [Section , "How to Create a Local Oracle Event Processing Server and Server Runtime"](#).

---

**Note:** If you use the Oracle Event Processing Visualizer to make changes to the Oracle Event Processing server `config.xml` (for example, editing a data source), you may overwrite `config.xml` file changes made manually. For more information, see [Section , "How to Configure Domain \(Runtime\) Settings for Oracle Event Processing Server"](#).

---

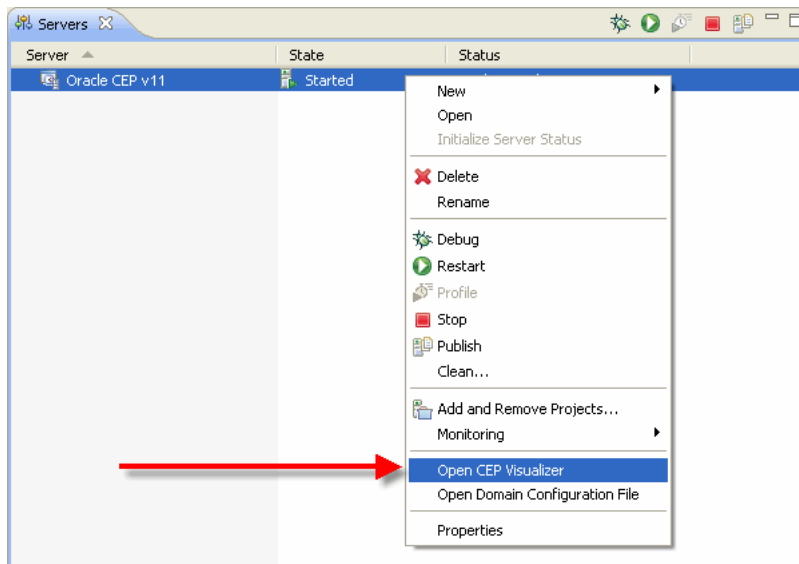
**To start the Oracle Event Processing Visualizer from Oracle Event Processing IDE for Eclipse:**

1. Start the server (see [Section , "How to Start a Local Oracle Event Processing Server"](#)).

Note that if you stop the server while Oracle Event Processing Visualizer is running, you might see informational messages about the service proxy. These are typically not errors.

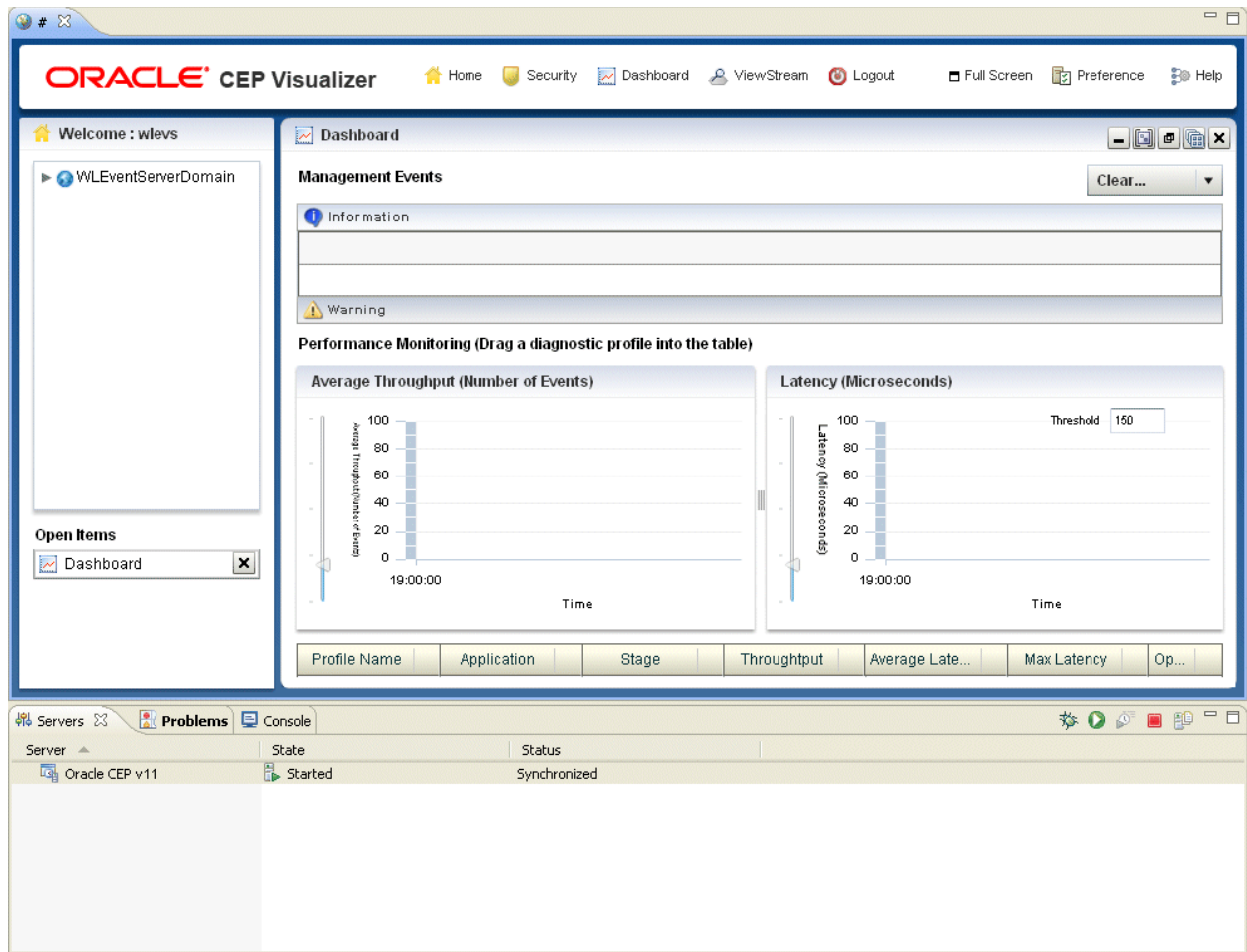
2. Right-click the running server in the Servers view and select **Open Oracle Event Processing Visualizer** as shown in [Figure 6–29](#).

**Figure 6–29** *Opening the Oracle Event Processing Visualizer*



The Oracle Event Processing Visualizer opens as shown in [Figure 6–30](#).

Figure 6–30 Oracle Event Processing Visualizer



- Use the Oracle Event Processing Visualizer as the *Oracle Fusion Middleware Visualizer User's Guide for Oracle Event Processing* describes.

## Debugging an Oracle Event Processing Application Running on an Oracle Event Processing Server

Because Oracle Event Processing applications are Java applications, standard Java debugging tools including those provided in Eclipse can be used with these applications.

This section describes:

- Section, "How to Debug an Oracle Event Processing Application Running on an Oracle Event Processing Server"

You can also use the load generator and csvgen adapter to simulate data feeds for testing. For more information, see [Chapter 21, "Testing Applications With the Load Generator and csvgen Adapter"](#).

## How to Debug an Oracle Event Processing Application Running on an Oracle Event Processing Server

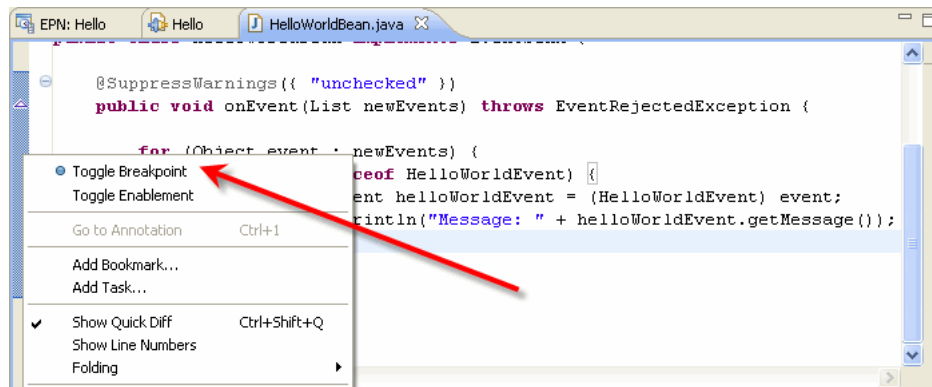
This section describes how to debug an Oracle Event Processing application running on an Oracle Event Processing server.

### To debug an Oracle Event Processing application running on an Oracle Event Processing server:

1. Set a breakpoint in the Java code you wish to debug.

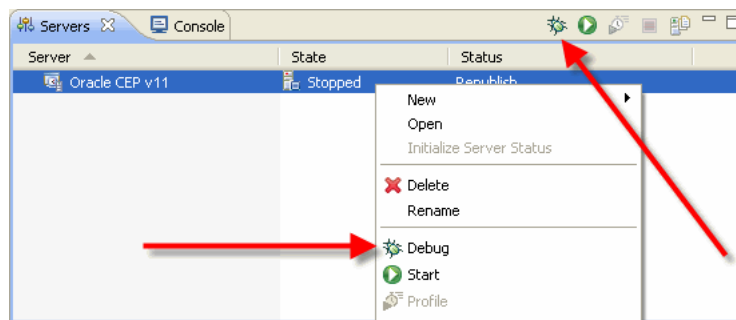
In this case, set the breakpoint by right-clicking in the gutter of the editor and selecting **Toggle Breakpoint** as [Figure 6–31](#) shows.

**Figure 6–31** 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 6–32](#):
  - 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 6–32** Starting the Oracle Event Processing Server in Debug Mode



4. The server will start, and when it gets to your breakpoint the thread will stop.  
If the Oracle Event Processing 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 Event Processing 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 , "How to Stop a Local Oracle Event Processing Server"](#)).





---

---

# Oracle Event Processing IDE for Eclipse and the Event Processing Network

This chapter describes how to use the Oracle Event Processing IDE for Eclipse to develop event processing networks (EPNs), where application components are wired together. The EPN Editor provides a graphical view of the EPN and offers visualization and navigation features to help you build Oracle Event Processing applications.

This chapter includes the following sections:

- [Opening the EPN Editor](#)
- [EPN Editor Overview](#)
- [Navigating the EPN Editor](#)
- [Using the EPN Editor](#)

## Opening the EPN Editor

You can open the EPN Editor from either the project folder or a context or configuration file of an Oracle Event Processing application.

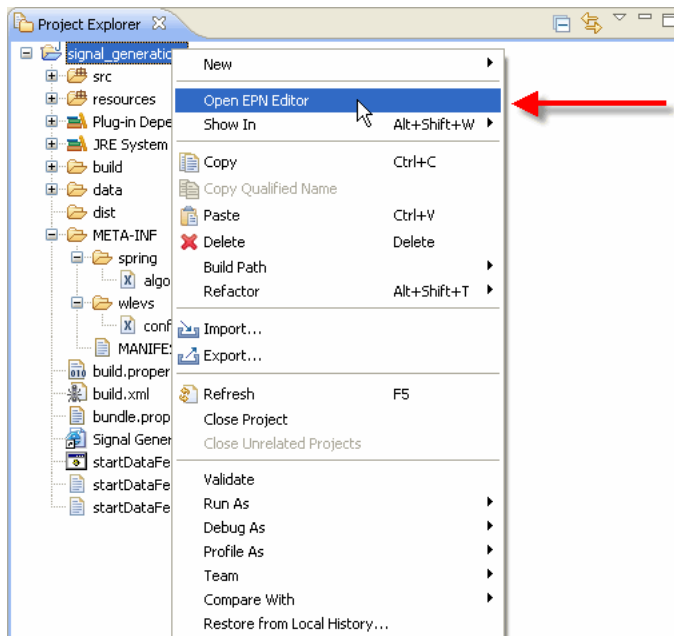
### How to Open the EPN Editor from a Project Folder

You can open the EPN Editor from the Eclipse project folder of an Oracle Event Processing application. Alternatively, you can open the EPN Editor from a context or configuration file (see [Section , "How to Open the EPN Editor from a Context or Configuration File"](#)).

#### To open the EPN Editor from a project:

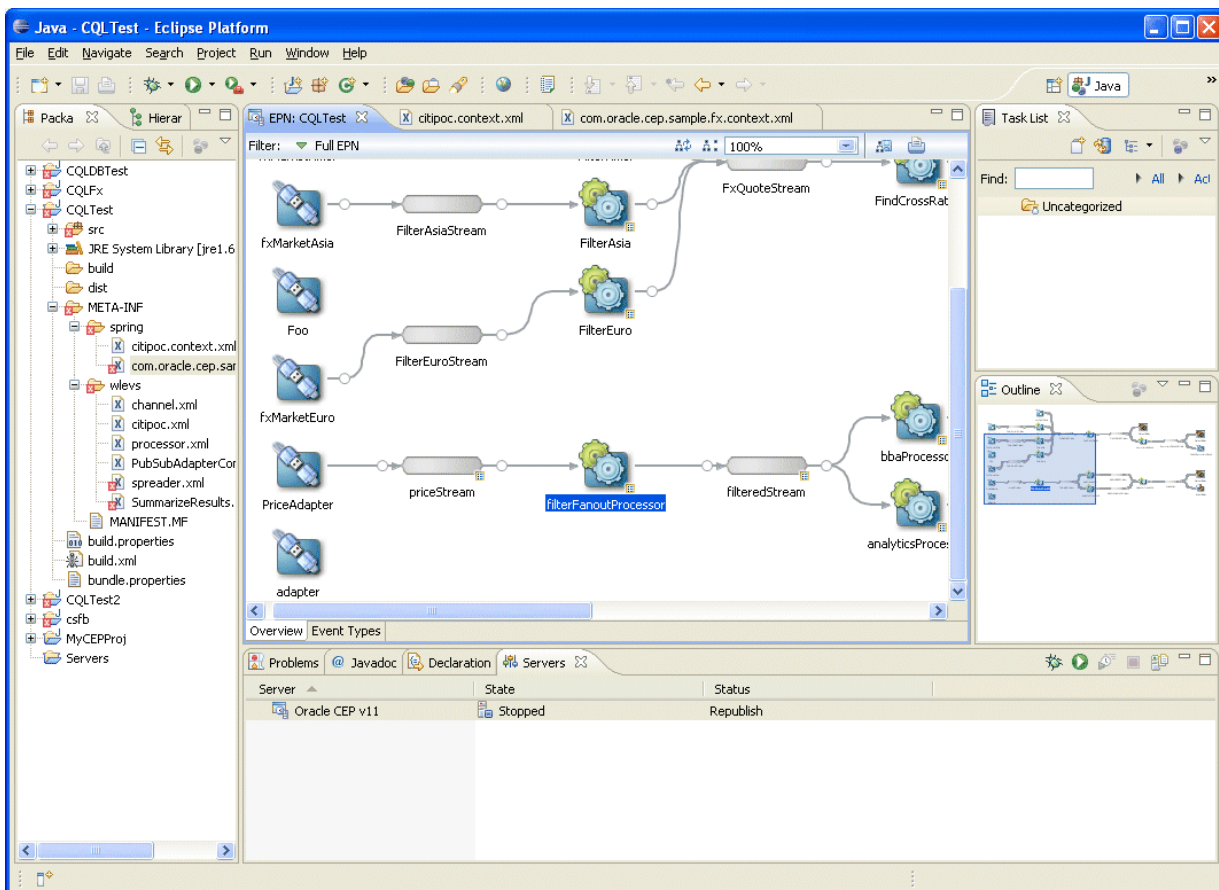
1. Launch the Oracle Event Processing IDE for Eclipse.
2. Open your Oracle Event Processing project in the Project Explorer.
3. Right-click the project folder and select **Open EPN Editor** as [Figure 7-1](#) shows.

**Figure 7-1** 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 Event Processing project, as Figure 7-2 shows.

**Figure 7-2** EPN Editor



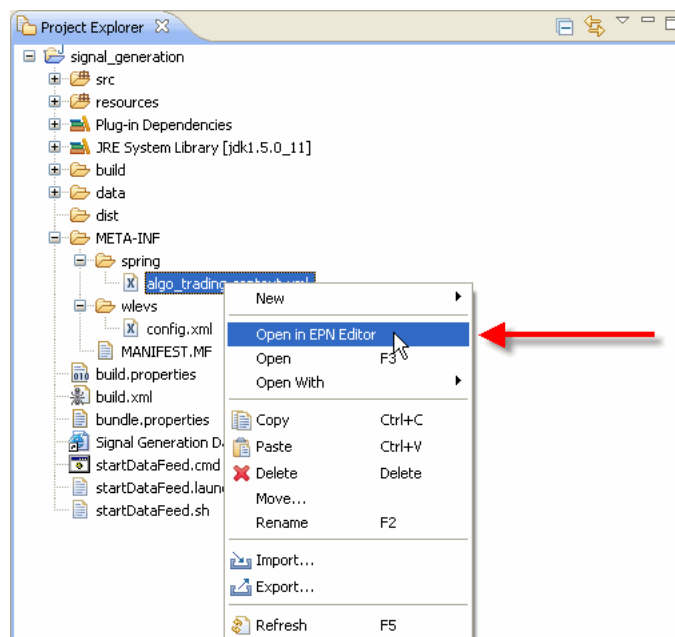
## How to Open the EPN Editor from a Context or Configuration File

You can open the EPN Editor from a Spring context file or an Oracle Event Processing server configuration file in an Oracle Event Processing application. Alternatively, you can open the EPN Editor from a context or configuration file (see [Section , "How to Open the EPN Editor from a Project Folder"](#))

### To open the EPN Editor from a context or configuration file:

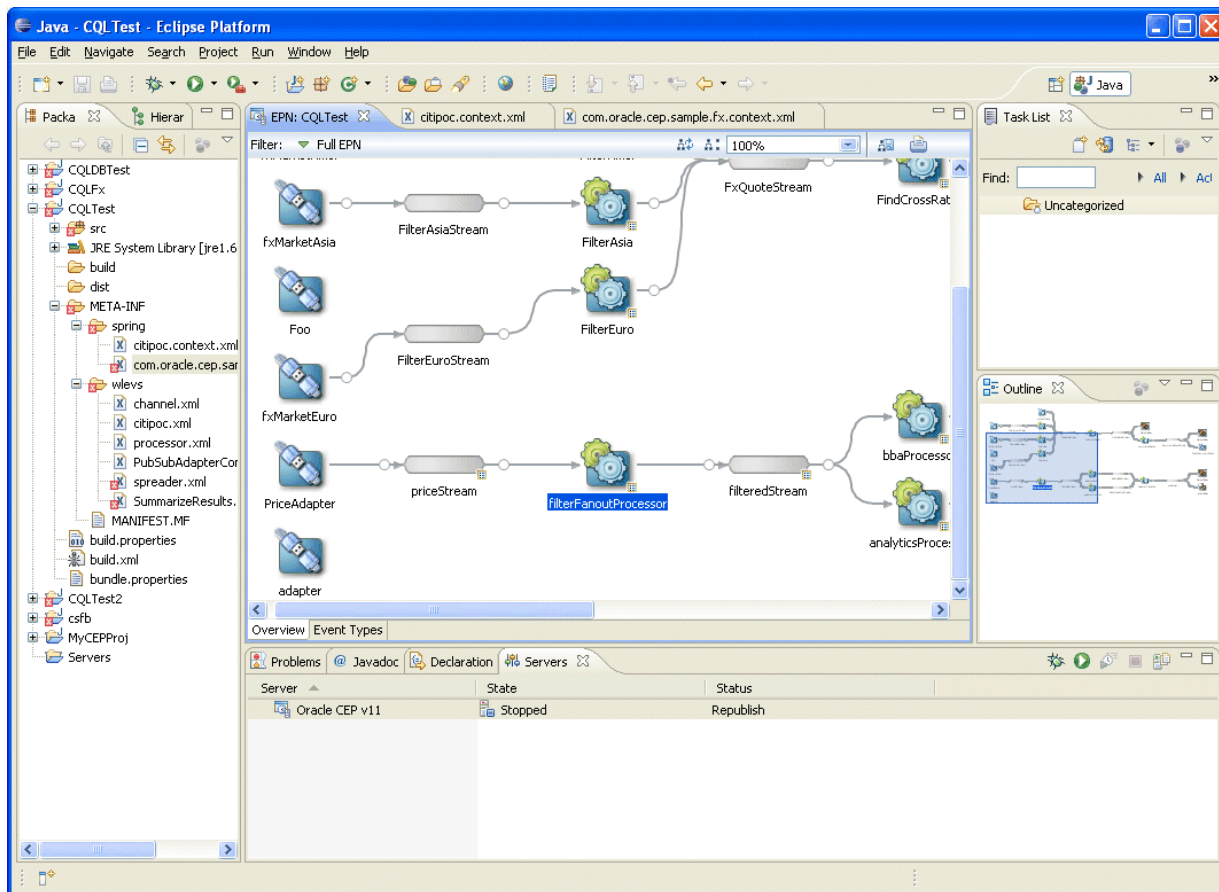
1. Launch the Oracle Event Processing IDE for Eclipse.
2. Open your Oracle Event Processing project in the Project Explorer.
3. Right-click a context or configuration file and select **Open in EPN Editor** as [Figure 7-3](#) shows.

**Figure 7-3** 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 Event Processing project, as [Figure 7-4](#) shows.

Figure 7-4 EPN Editor



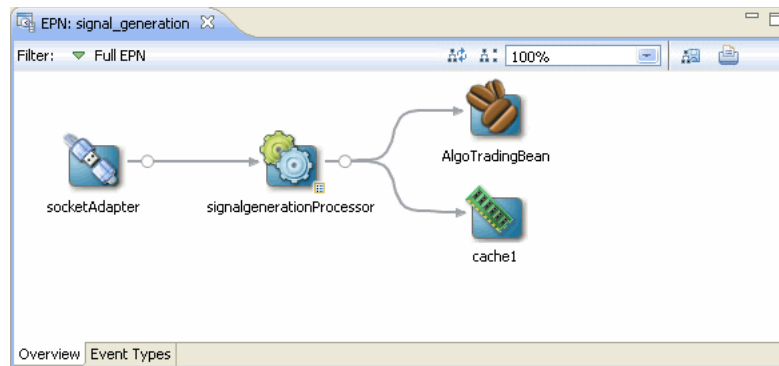
## EPN Editor Overview

This section describes the main controls you use to manage the EPN view and how the EPN Editor displays Oracle Event Processing application information, including:

- [Section , "Flow Representation"](#)
- [Section , "Filtering"](#)
- [Section , "Zooming"](#)
- [Section , "Layout"](#)
- [Section , "Showing and Hiding Unconnected Beans"](#)
- [Section , "Printing and Exporting to an Image"](#)
- [Section , "Configuration Badging"](#)
- [Section , "Link Specification Location Indicator"](#)
- [Section , "Nested Stages"](#)
- [Section , "Event Type Repository Editor"](#)

## Flow Representation

The primary display in the editor is of the flow inside the application as [Figure 7-5](#) shows.

**Figure 7-5 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:

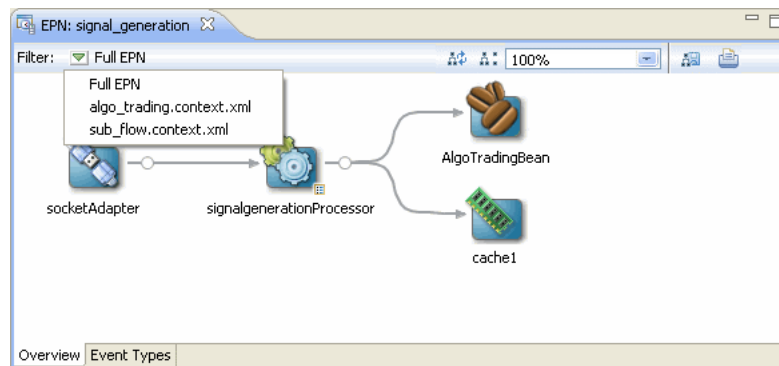
- [Section , "Configuration Badging"](#)
- [Section , "Link Specification Location Indicator"](#)

## Filtering

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 EPN for a single Oracle Event Processing application bundle 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 7-6](#) shows.

**Figure 7-6 Filtering the EPN by Assembly File**

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.

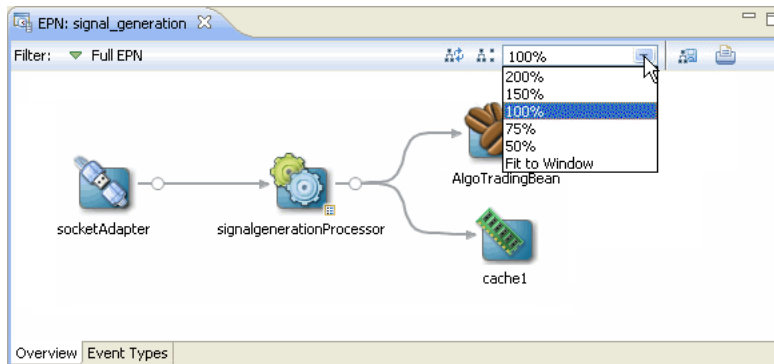
If the assembly file the EPN Editor edits is not open in an Eclipse source editor, then the edits are immediately applied to the file on disk.

For more information, see [Section , "Creating EPN Assembly Files"](#).

## Zooming

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 7-7](#) shows. To fit the EPN into the current EPN Editor window, select **Fit to Window**.

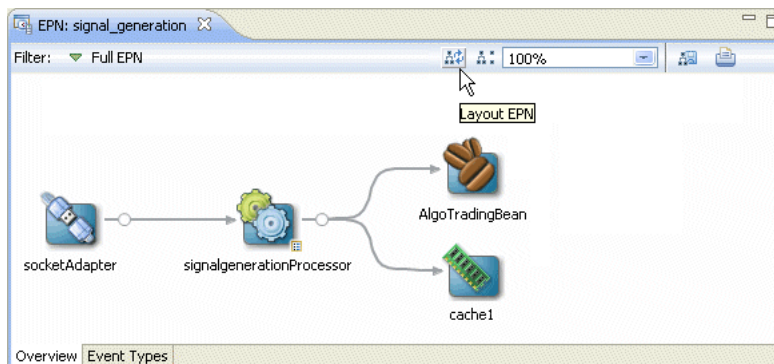
**Figure 7-7 Zoom Level**



## Layout

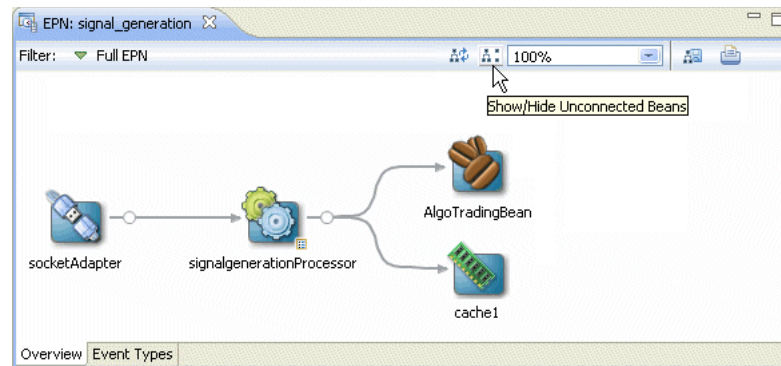
You can optimize and simplify the EPN layout by clicking **Layout EPN** as [Figure 7-8](#) shows.

**Figure 7-8 Optimize Layout**



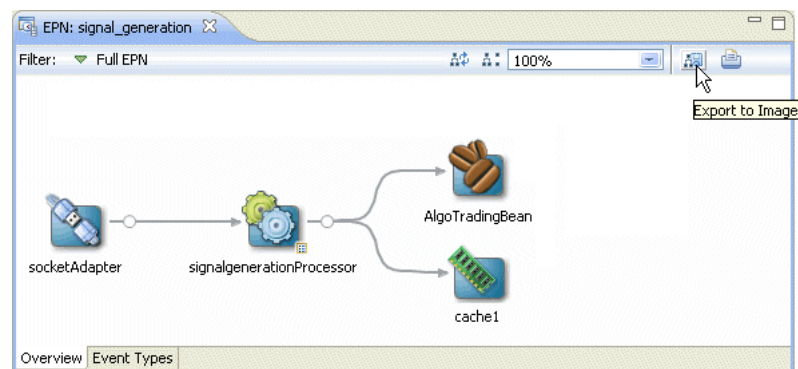
## Showing and Hiding Unconnected Beans

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 7-9](#) shows. For more information, see [Section , "Laying Out Nodes"](#).

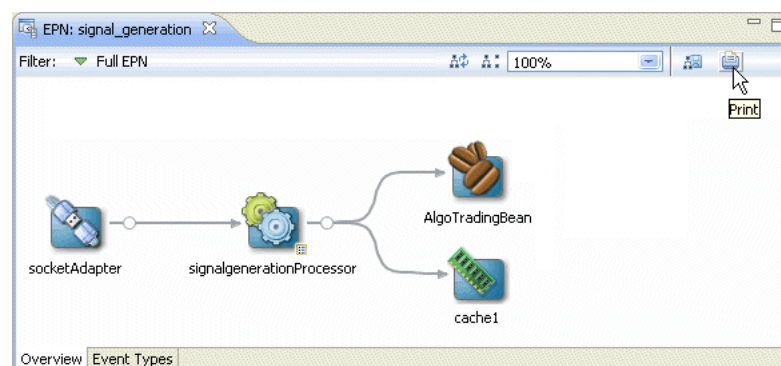
**Figure 7–9 Show/Hide Unconnected Beans**

## Printing and Exporting to an Image

You can export the EPN Editor view to an image file by clicking **Export to Image** as [Figure 7–10](#) shows. You can export the image as a .bmp, .gif, .jpg, or .png file.

**Figure 7–10 Exporting the EPN as an Image File**

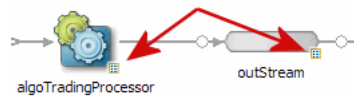
You can print the EPN Editor view by clicking **Print** as [Figure 7–11](#) shows.

**Figure 7–11 Printing the EPN**

## 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 7–12](#) shows.

**Figure 7–12 Configuration Badging**



Nodes with this badge will also have the **Go To Configuration Source** context menu item.

## 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 7–13 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 7–14 shows the corresponding assembly file.

**Figure 7–13 Link Source**



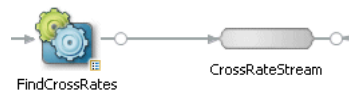
**Figure 7–14 Link Source Assembly File**

```
<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 7–15 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 7–16 shows the corresponding assembly file.

**Figure 7–15 Link Listener**





**Figure 7–16 Link Listener Assembly File**

```

<wlevs:processor id="FindCrossRates" provider="cql">
  <wlevs:listener ref="CrossRateStream"/>
</wlevs:processor>

<wlevs:channel id="CrossRateStream" event-type="SpreaderOutputEvent" advertise="true">
  <wlevs:listener ref="summarizeResults"/>
  <wlevs:listener>
    <bean class="com.oracle.cep.sample.fx.OutputBean" autowire="byName"/>
  </wlevs:listener>
</wlevs:channel>

```

## 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 , "Connecting Nodes"](#).

Consider the EPN that [Figure 7–17](#) shows. [Example 7–1](#) 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 7–17 EPN With Nested Bean****Example 7–1 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 7–18](#) shows. [Example 7–2](#) 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 7–18 EPN With all Nodes Nested**



**Example 7–2 Assembly Source for EPN With all Nodes Nested**

```

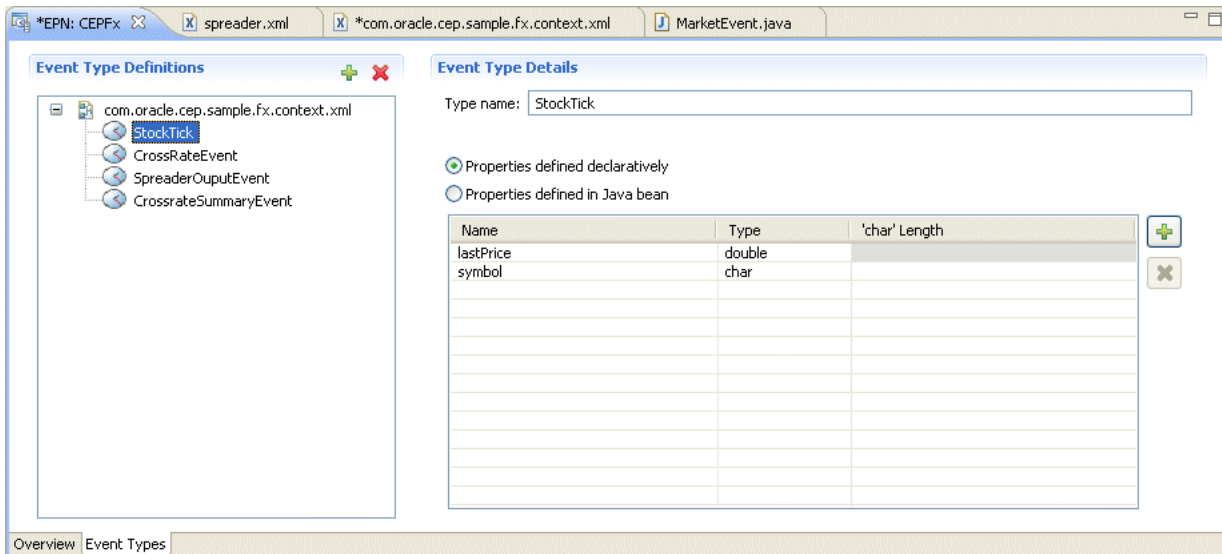
<wlevs:adapter id="helloworldAdapter" class="com.bea.wlevs.adapter.example.helloworld.HelloWorldAdapter" >
  <wlevs:instance-property name="message" value="HelloWorld - the current time is:"/>
  <wlevs:listener>
    <wlevs:channel id="helloworldInputChannel" event-type="HelloWorldEvent" >
      <wlevs:listener>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

## Event Type Repository Editor

You can create and edit JavaBean and tuple event types using the event type repository editor.

To open the event type repository editor, click on the **Event Types** tab in the EPN editor as [Figure 7–19](#) shows.

**Figure 7–19 Event Type Repository Editor**



For more information, see:

- [Section , "How to Create an Oracle Event Processing Event Type as a JavaBean Using the Event Type Repository Editor"](#)
- [Section , "How to Create an Oracle Event Processing Event Type as a Tuple Using the Event Type Repository Editor"](#)

For information on the other types of events you can create, see [Section , "Overview of Oracle Event Processing Event Types"](#).

## 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 Event Processing application. Oracle Event Processing IDE for Eclipse offers the following features to help navigate the EPN Editor:

- [Section , "Moving the Canvas"](#)
- [Section , "Shortcuts to Component Configuration and EPN Assembly Files"](#)
- [Section , "Hyperlinking"](#)
- [Section , "Context Menus"](#)
- [Section , "Browsing Oracle Event Processing Types"](#)

## 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.

## 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 7-20](#).

**Figure 7-20 Node with Configuration Badge**



For more information, see:

- [Section , "Configuration Badging"](#)
- [Section , "Hyperlinking"](#)

## Hyperlinking

When editing a component configuration file, EPN assembly file, or Oracle CQL statement, 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.

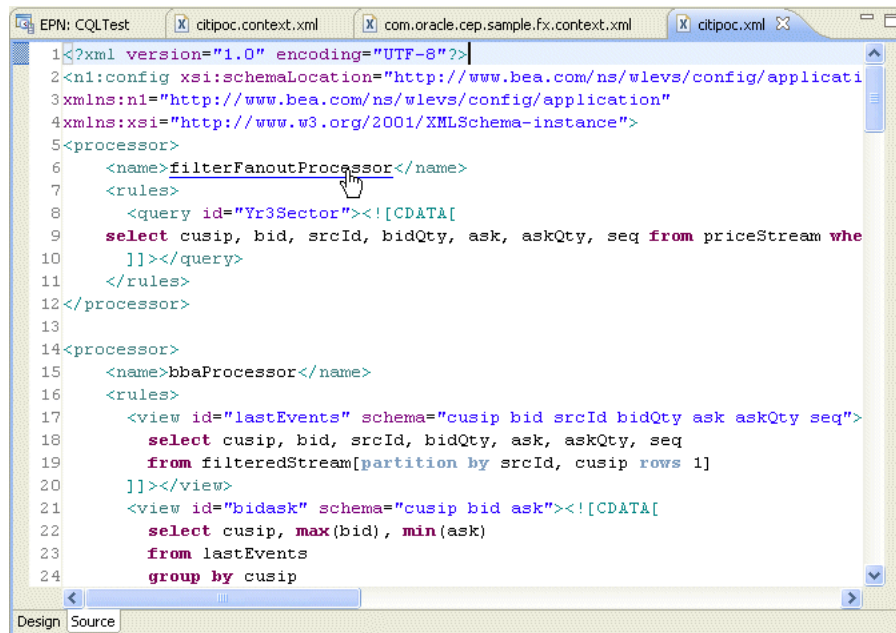
This section describes:

- [Section , "Hyperlinking in Component Configuration and EPN Assembly Files"](#)
- [Section , "Hyperlinking in Oracle CQL Statements"](#)

### Hyperlinking in Component Configuration and EPN Assembly Files

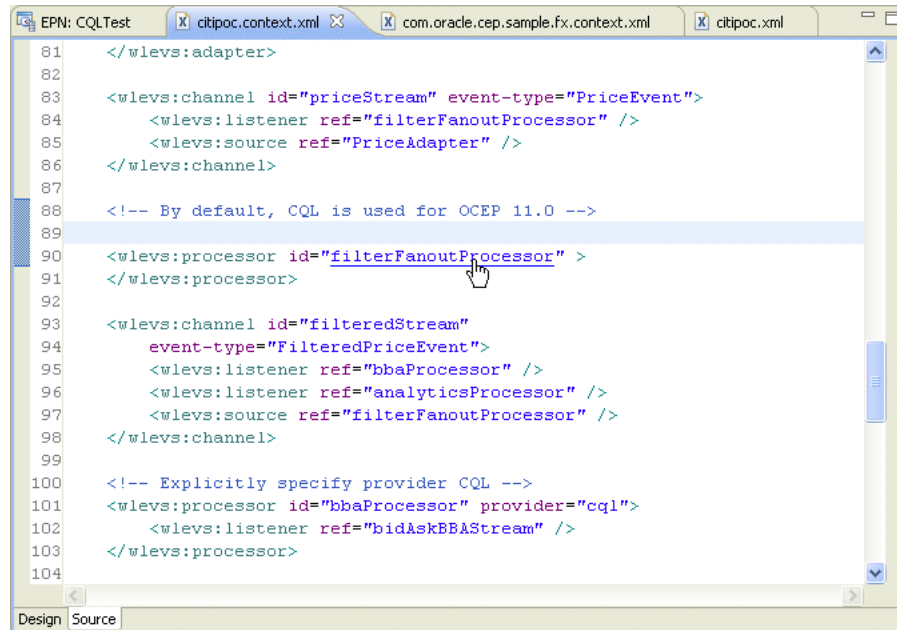
Figure 7–21 shows a component configuration file with the cursor over the value of a processor element name child element while holding down the **Ctrl** key. The name value has an underline to indicate it is a hyperlink. Click this link to jump to the corresponding element in the EPN assembly file as Figure 7–22 shows.

**Figure 7–21 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 7–22 EPN Assembly File: Hyperlinking to Component Configuration File**



### Hyperlinking in Oracle CQL Statements

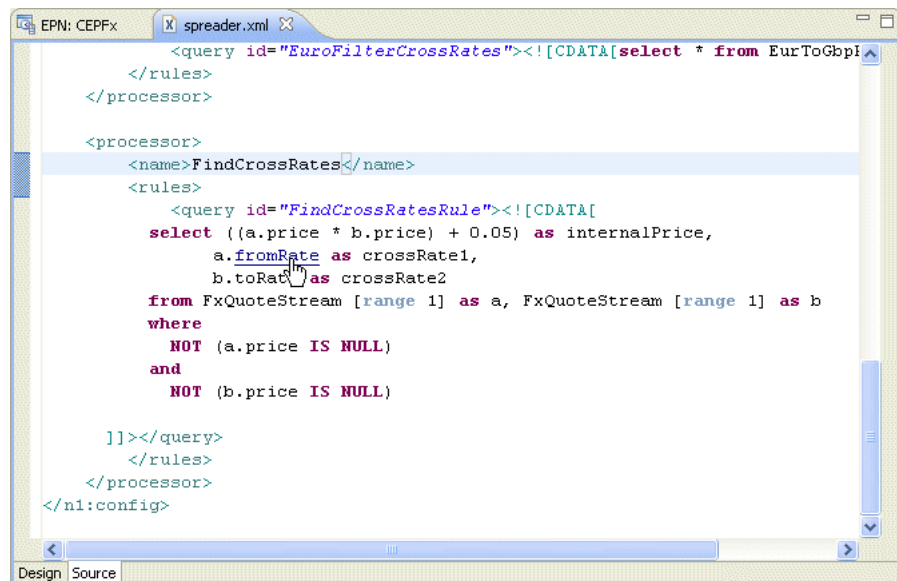
Figure 7–23 shows a component configuration file with the cursor over an event attribute while holding down the **Ctrl** key. The `fromRate` attribute has an underline to indicate it is a hyperlink. Click this link to jump to the corresponding event definition in the EPN assembly file as Figure 7–24 shows.

---

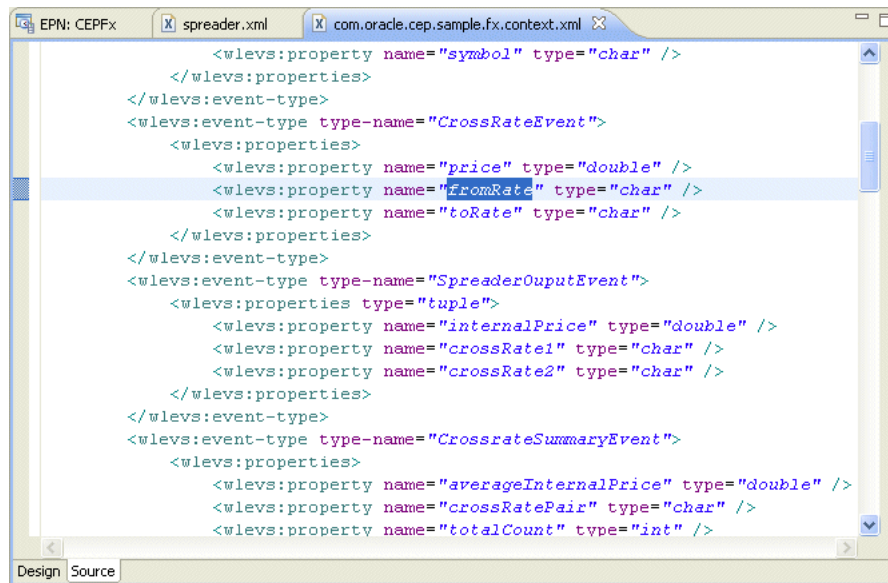
**Note:** Hyperlinking in Oracle SQL statements is designed for simple use cases and may not work as expected in more complex implementations.

---

**Figure 7–23 Oracle CQL Statement: Event Schema**



**Figure 7–24 Corresponding Event Definition in EPN Assembly File**



Similarly, you can **Ctrl-click** the FxQuoteStream channel in the Oracle CQL statement that [Figure 7–23](#) shows to jump to the channel’s definition. This is applicable wherever references to external objects are present in a Oracle CQL statement.

## 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 , "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 , "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.

## Browsing Oracle Event Processing Types

A typical Oracle Event Processing project contains many instances of Oracle Event Processing types such as adapters, channels, processors, event beans. In a large, complex Oracle Event Processing project, it can be a challenge to locate a particular instance. The Oracle Event Processing IDE for Eclipse provides an Oracle Event Processing type browser that you can use to quickly locate instances of any Oracle Event Processing type.

### How to Browse Oracle Event Processing Types

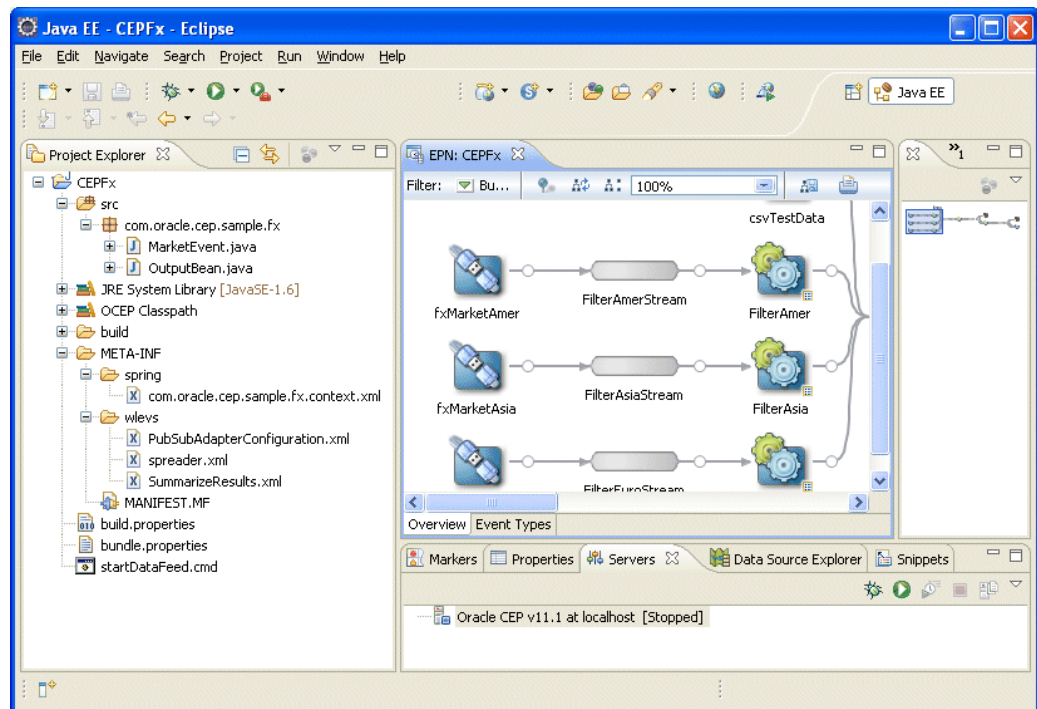
You can open the Oracle Event Processing type browser using the keyboard short cut **Ctrl-Alt-T**.

#### To browse Oracle Event Processing types:

1. Open an Oracle Event Processing project.

In the following procedure, consider the Oracle Event Processing project that [Figure 7–25](#) shows. This is based on the Oracle Event Processing foreign exchange example. For more information on this example, see [Section , "Foreign Exchange \(FX\) Example"](#).

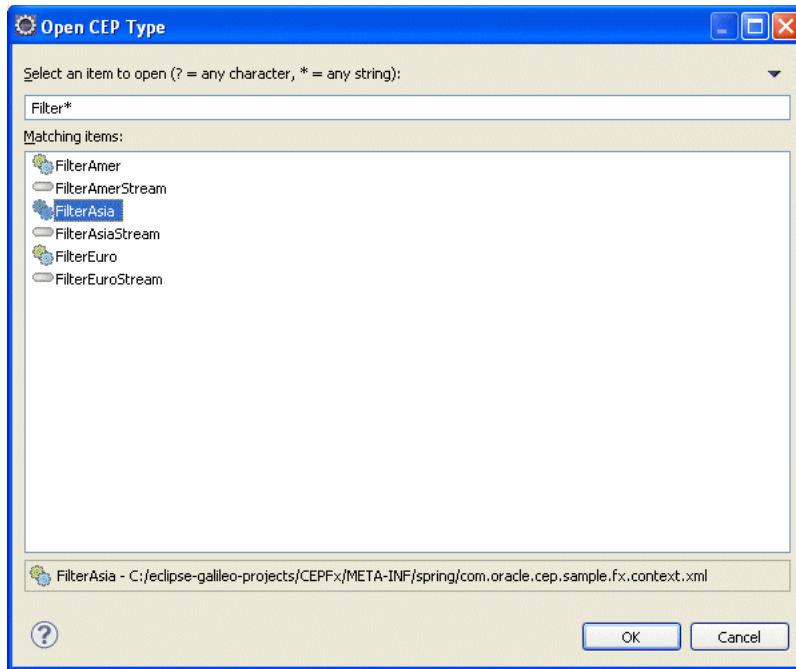
**Figure 7–25 Example Oracle Event Processing EPN**



2. Type the keyboard short cut **Ctrl-Alt-T**.

The Oracle Event Processing type browser appears as [Figure 7–26](#) shows.

**Figure 7–26 Oracle Event Processing Type Browser**



3. Configure the Oracle Event Processing Type dialog as shown in [Table 7–1](#).

**Table 7–1 Oracle Event Processing Type Dialog**

Attribute	Description
Select an item to open	Specify a filter to match the names of the items you want to find. Use the ? wildcard for any single character and the * wildcard for any string of two or more characters.
Matching items	The list of Oracle Event Processing type instances whose name matches the filter you specified.

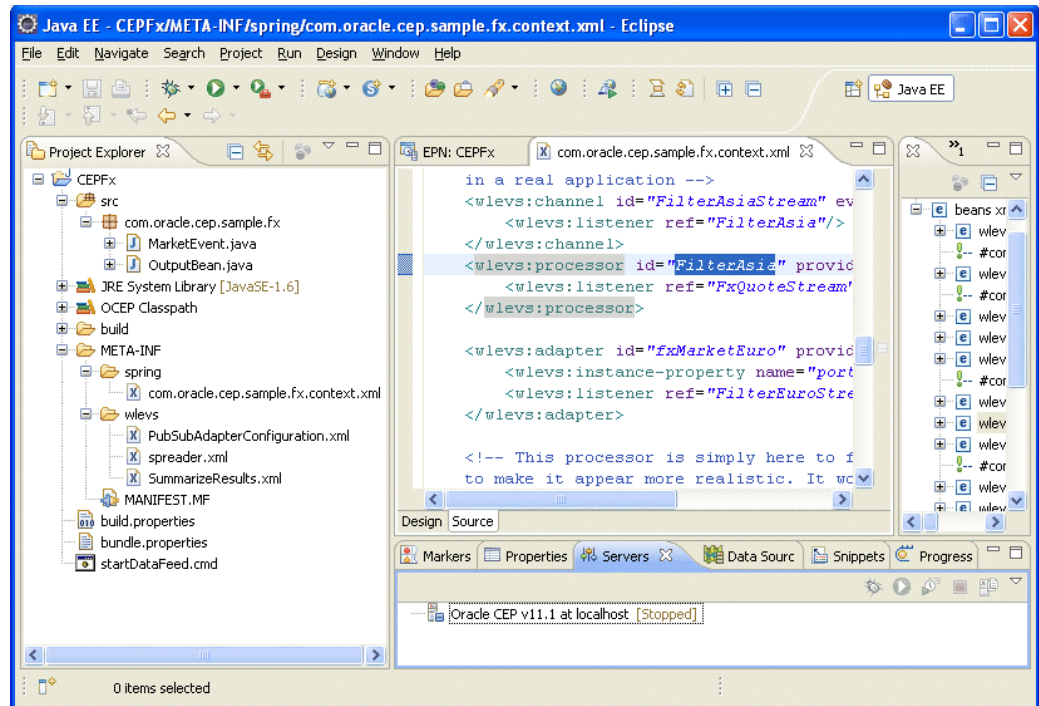
By default, the status line below the Matching items list shows the fully qualified path to the selected item in the Select an item to open list. To toggle status line display, click on the pull-down menu in the right hand corner and select **Show Status Line**.

4. Select a type in the Matching Items list and click **OK**.

The type is opened in the source file in which it is defined. For example, selecting **FilterAsia** from the **Matching Items** list and clicking **OK** opens the `com.oracle.cep.sample.fx.content.xml` EPN assembly file in which this processor is defined as [Figure 7–27](#) shows.

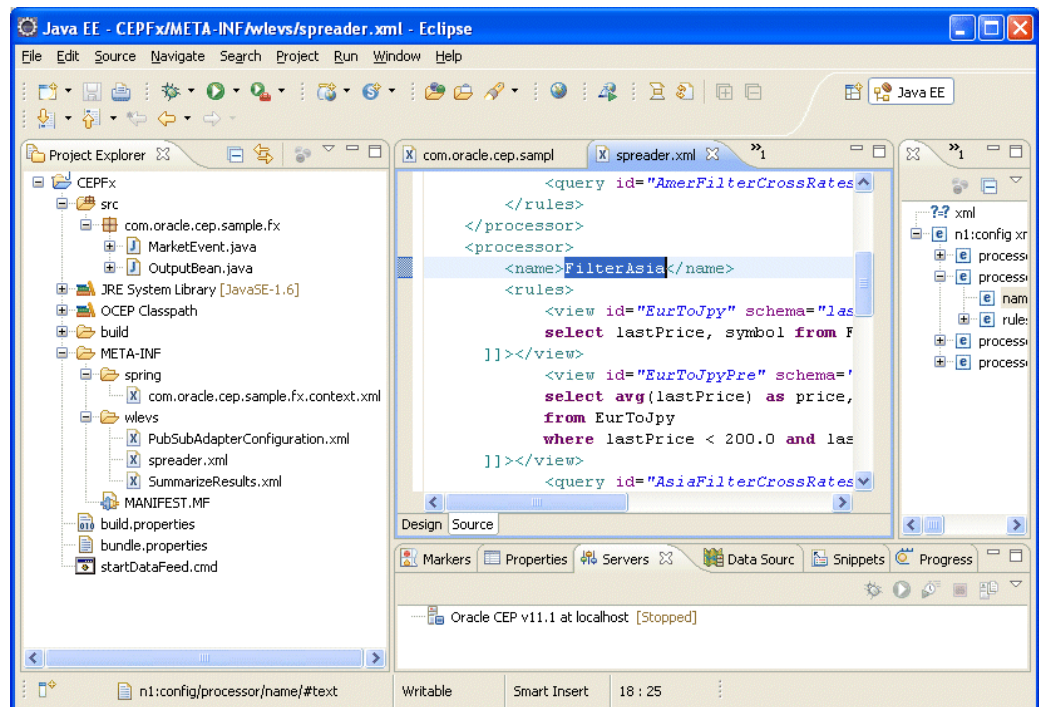


**Figure 7–27** Opening the FilterAsia EPN Assembly File



To navigate to the corresponding component configuration file as [Figure 7–28](#) shows, **Ctrl-click** the **FilterAsia** id attribute value.

**Figure 7–28** Opening the FilterAsia Component Configuration File



For more information on hyperlinking, see [Section , "Hyperlinking"](#).

## 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. You can use a single EPN assembly file or multiple EPN assembly files (for more information, see [Section , "Filtering"](#)).

The following sections describe EPN Editor editing tasks, including:

- [Section , "Creating Nodes"](#)
- [Section , "Connecting Nodes"](#)
- [Section , "Laying Out Nodes"](#)
- [Section , "Renaming Nodes"](#)
- [Section , "Deleting Nodes"](#)




For more information, see:

- [Section , "Oracle Event Processing Project Overview"](#)
- [Section , "EPN Editor Overview"](#)
- [Section , "Opening the EPN Editor"](#)
- [Section , "Navigating the EPN Editor"](#)





## 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 7-2](#) lists.

**Table 7-2 EPN Editor Icons**

Node	Description
	<p><b>Adapter:</b> a node that interfaces an event data source with the EPN or interfaces the EPN with an event data sink.</p> <p>For more information, see:</p> <ul style="list-style-type: none"> <li>■ <a href="#">Section , "How to Create an Adapter Node"</a></li> <li>■ <a href="#">Chapter 11, "Integrating the Java Message Service"</a></li> <li>■ <a href="#">Chapter 12, "Integrating an HTTP Publish-Subscribe Server"</a></li> <li>■ <a href="#">Chapter 15, "Integrating an External Component Using a Custom Adapter"</a></li> <li>■ <a href="#">Chapter 21, "Testing Applications With the Load Generator and csvgen Adapter"</a></li> </ul>
	<p><b>Channel:</b> a node that conveys events between an event data source and an event data sink.</p> <p>For more information, see:</p> <ul style="list-style-type: none"> <li>■ <a href="#">Section , "How to Create a Basic Node"</a></li> <li>■ <a href="#">Chapter 10, "Connecting EPN Stages Using Channels"</a></li> </ul>
	<p><b>Processor:</b> a node that executes Oracle CQL or EPL rules on the event data offered to it by one or more channels.</p> <p>For more information, see:</p> <ul style="list-style-type: none"> <li>■ <a href="#">Section , "How to Create a Processor Node"</a></li> <li>■ <a href="#">Chapter 17, "Querying an Event Stream with Oracle CQL"</a></li> <li>■ <a href="#">Chapter 19, "Querying an Event Stream with Oracle EPL"</a></li> </ul>

**Table 7–2 (Cont.) EPN Editor Icons**

Node	Description
	<p><b>Event Bean:</b> a node similar to a standard Spring bean except that it can be managed by the Oracle Event Processing management framework and can actively use the capabilities of the Oracle Event Processing server container.</p> <p>For more information, see:</p> <ul style="list-style-type: none"> <li>▪ <a href="#">Section , "How to Create a Basic Node"</a></li> <li>▪ <a href="#">Chapter 16, "Handling Events with Java"</a></li> </ul>
	<p><b>Spring Bean:</b> a Plain Old Java Object (POJO) node that consumes events. A Spring bean is managed by the Spring framework.</p> <p>For more information, see:</p> <ul style="list-style-type: none"> <li>▪ <a href="#">Section , "How to Create a Basic Node"</a></li> <li>▪ <a href="#">Chapter 16, "Handling Events with Java"</a></li> </ul>
	<p><b>Cache:</b> a node that provides a temporary storage area for events, created exclusively to improve the overall performance of your Oracle Event Processing application.</p> <p>For more information, see:</p> <ul style="list-style-type: none"> <li>▪ <a href="#">Section , "How to Create a Basic Node"</a></li> <li>▪ <a href="#">Chapter 13, "Integrating a Cache"</a></li> </ul>
	<p><b>Table:</b> a node that connects a relational database table to the EPN as an event data source.</p> <p>For more information, see:</p> <ul style="list-style-type: none"> <li>▪ <a href="#">Section , "How to Create a Basic Node"</a></li> <li>▪ <a href="#">Section , "Configuring an Oracle CQL Processor Table Source"</a></li> </ul>

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 , "Laying Out Nodes"](#).

When a child node is nested within a parent node, its icon appears within a box. For more information, see [Section , "Nested Stages"](#).

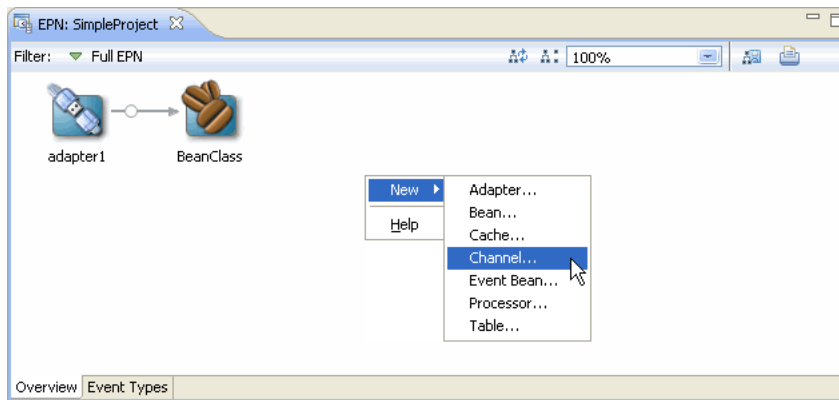
### How to Create a Basic Node

Basic nodes include beans, caches, channels, event beans, and tables.

For information on how to create other nodes, see [Section , "Creating Nodes"](#).

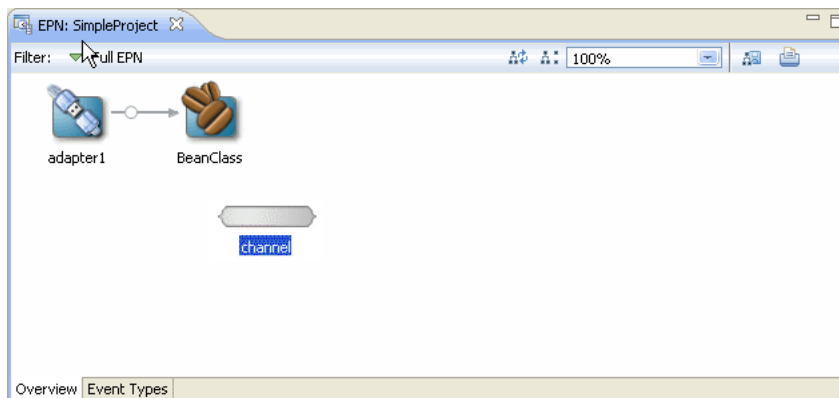
#### To create a basic node:

1. Open the EPN Editor (see [Section , "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 7–29](#) shows.

**Figure 7–29 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 7–30](#) shows.

**Figure 7–30 New Basic Node**

To rename the node, see [Section , "Renaming Nodes"](#).

To reposition the node and update the EPN Editor layout, see [Section , "Laying Out Nodes"](#).

4. Optionally, configure additional node options.

See:

- [Chapter 10, "Connecting EPN Stages Using Channels"](#)
- [Section , "Configuring an Oracle CQL Processor Table Source"](#)
- [Chapter 13, "Integrating a Cache"](#)
- [Chapter 15, "Integrating an External Component Using a Custom Adapter"](#)
- [Chapter 16, "Handling Events with Java"](#)
- [Chapter 21, "Testing Applications With the Load Generator and csvgen Adapter"](#)

## How to Create an Adapter Node

This section describes how to create an adapter using the EPN Editor, including:

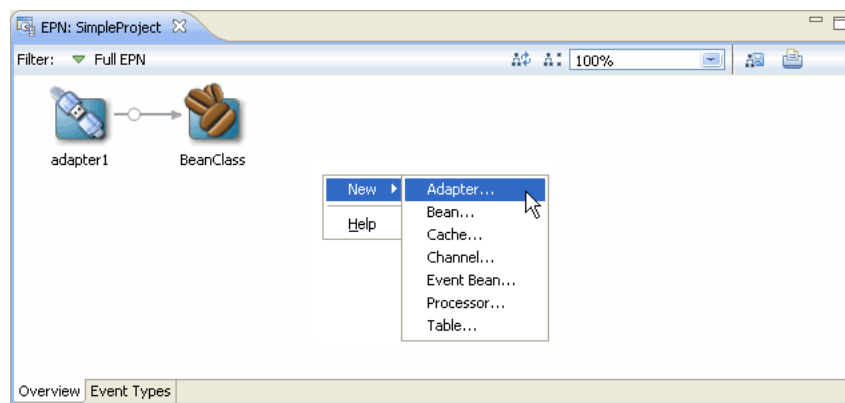
- JMS adapters (in-bound or out-bound)
- HTTP publish-subscribe server adapters (publishing or subscribing)

For information on how to create other nodes, see [Section , "Creating Nodes"](#).

### To create an adapter node:

1. Open the EPN Editor (see [Section , "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 7–31](#) shows.

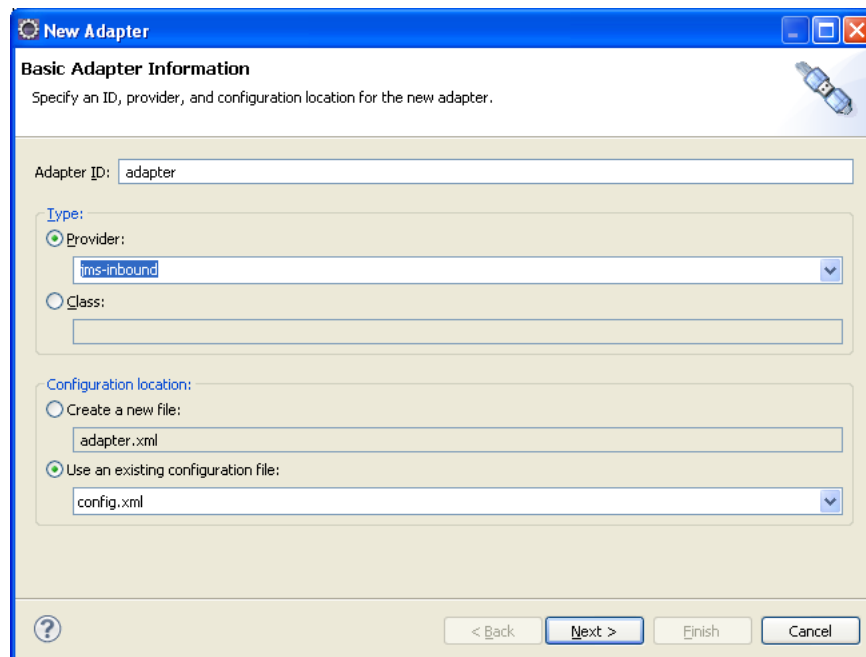
**Figure 7–31** *Creating an Adapter Node*



3. Select node type **Adapter**.

The New Adapter wizard appears as shown in [Figure 7–32](#).

**Figure 7–32** *New Adapter Wizard*



4. Configure the New Adapter Wizard - Page 1 as shown in [Table 7-3](#).

**Table 7-3 New Adapter Wizard - Page 1**

Attribute	Description
Adapter ID	Specifies the ID of the adapter EPN element and the name of the associated adapter configuration element.
Provider	Select the adapter provider type from the pull-down menu for an adapter already defined in the Oracle Event Processing component configuration schema. Select one of: <ul style="list-style-type: none"> <li>■ <b>jms-inbound</b>: JMS in-bound adapter.</li> <li>■ <b>jms-outbound</b>: JMS out-bound adapter.</li> <li>■ <b>httppub</b>: HTTP publish-subscribe adapter for publishing.</li> <li>■ <b>httpsub</b>: HTTP publish-subscribe adapter for subscribing.</li> </ul>
Class	Specify the fully qualified Java class name of a custom adapter.  <b>NOTE:</b> If you are using a custom adapter factory, you must add the <code>wlevs:factory</code> element manually. For more information, see <a href="#">Chapter 15, "Integrating an External Component Using a Custom Adapter"</a> .
Create a new file	Creates the adapter component configuration in a new file. The new file is created in the application's <code>META-INF/wlevs</code> directory with the same name as the adapter ID.
Use an existing configuration file	Creates the adapter component configuration in an existing configuration file. The new adapter configuration element is appended to the configurations in the selected file.

5. Proceed depending on how you configured the adapter implementation:
  - a. If you selected **Class**, Proceed to step 8.
  - b. If you selected **Provider**, proceed to step 6.
6. Click **Next**.  
The provider-specific New Adapter Wizard page appears.
7. Configure the provider-specific New Adapter Wizard page as the following figures show:
  - [Figure 7-33, "New Adapter Wizard - jms-inbound"](#)  
See [Section , "JMS Inbound Adapter Component Configuration"](#).
  - [Figure 7-34, "New Adapter Wizard - jms-outbound"](#)  
See [Section , "JMS Outbound Adapter Component Configuration"](#).
  - [Figure 7-35, "New Adapter Wizard - httppub"](#)  
See [Section , "HTTP Pub-Sub Adapter for Publishing Component Configuration"](#).
  - [Figure 7-36, "New Adapter Wizard - httpsub"](#)

See Section , "HTTP Pub-Sub Adapter for Subscribing Component Configuration".

**Figure 7–33 New Adapter Wizard - jms-inbound**

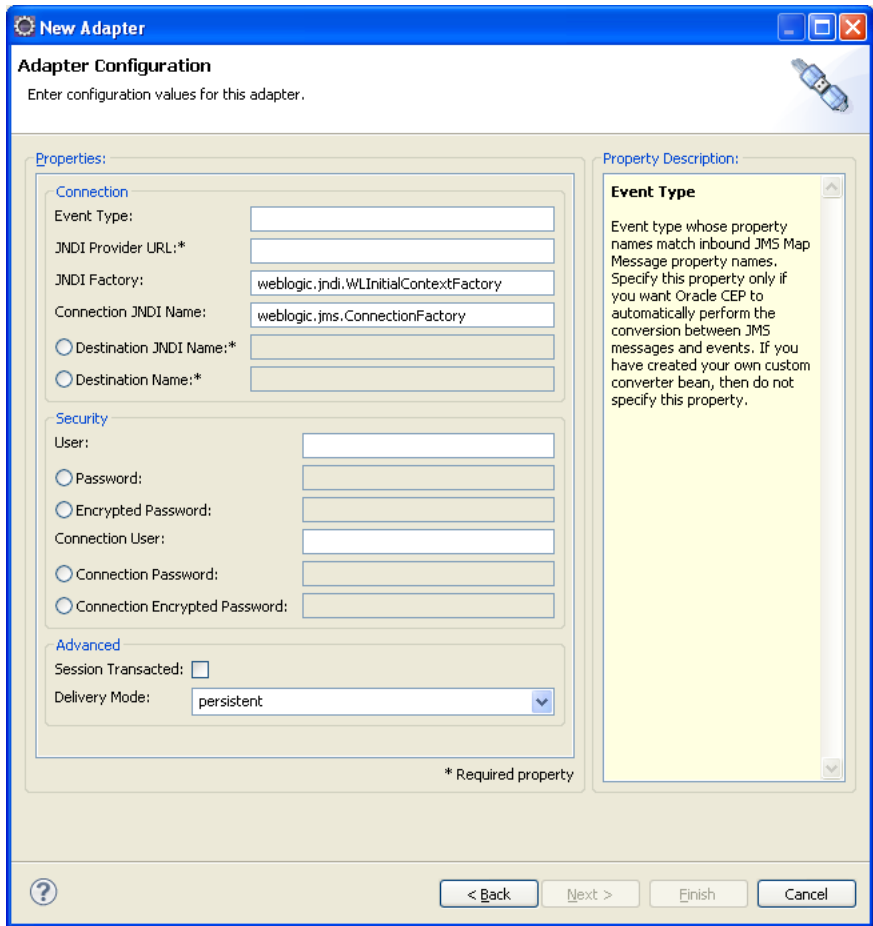
The screenshot shows the 'New Adapter' wizard window with the title 'Adapter Configuration'. The window contains the following sections:

- Properties:**
  - Connection:**
    - Event Type: [Text Field]
    - JNDI Provider URL:\* [Text Field]
    - JNDI Factory: weblogic.jndi.WLInitialContextFactory [Text Field]
    - Connection JNDI Name: weblogic.jms.ConnectionFactory [Text Field]
    - Destination JNDI Name:\* [Text Field]
    - Destination Name:\* [Text Field]
  - Security:**
    - User: [Text Field]
    - Password: [Text Field]
    - Encrypted Password: [Text Field]
    - Connection User: [Text Field]
    - Connection Password: [Text Field]
    - Connection Encrypted Password: [Text Field]
  - Advanced:**
    - Work Manager: [Text Field]
    - Concurrent Consumers: [Text Field]
    - Message Selector: [Text Field]
    - Session Acknowledgment Mode: AUTO\_ACKNOWLEDGE [Dropdown]
    - Session Transacted:
- Property Description:**
  - Event Type**  
Event type whose property names match inbound JMS Map Message property names. Specify this property 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 property.

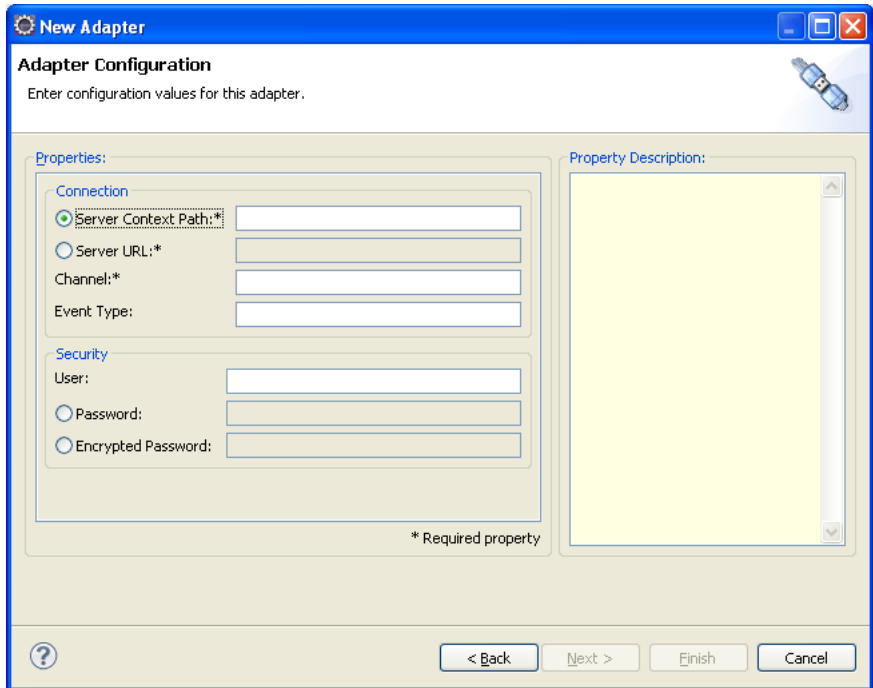
\* Required property

Navigation buttons: < Back, Next >, Finish, Cancel

**Figure 7-34 New Adapter Wizard - jms-outbound**



**Figure 7-35 New Adapter Wizard - httppub**

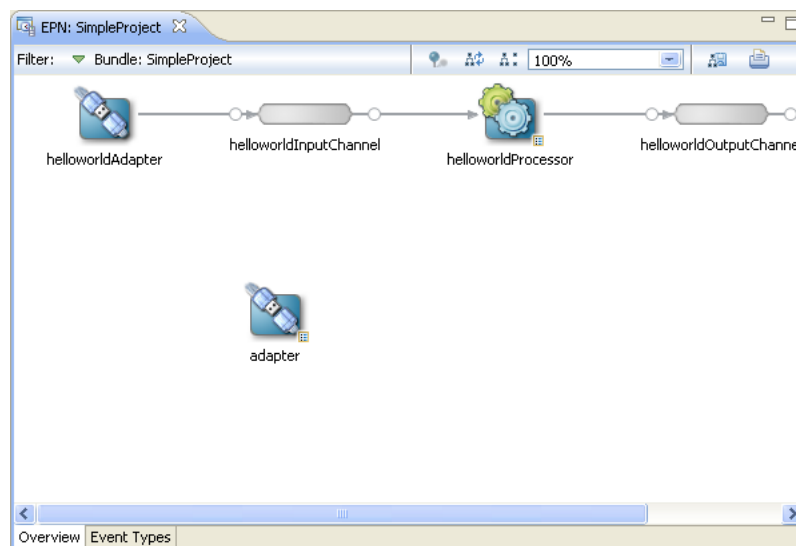




**Figure 7–36 New Adapter Wizard - https**

8. Click **Finish**.
9. Use the new adapter node on the EPN.

The EPN Editor creates the adapter configuration in the file you specified in the New Adapter wizard, edits the source file indicated in the EPN Editor filter, and displays the new EPN node as [Figure 7–37](#) shows.

**Figure 7–37 New Adapter Node**

To rename the node, see [Section , "Renaming Nodes"](#).

To reposition the node and update the EPN Editor layout, see [Section , "Laying Out Nodes"](#).

10. Optionally, configure additional node options.

For more information, see:

- [Chapter 11, "Integrating the Java Message Service"](#)
- [Chapter 12, "Integrating an HTTP Publish-Subscribe Server"](#)

### How to Create a Processor Node

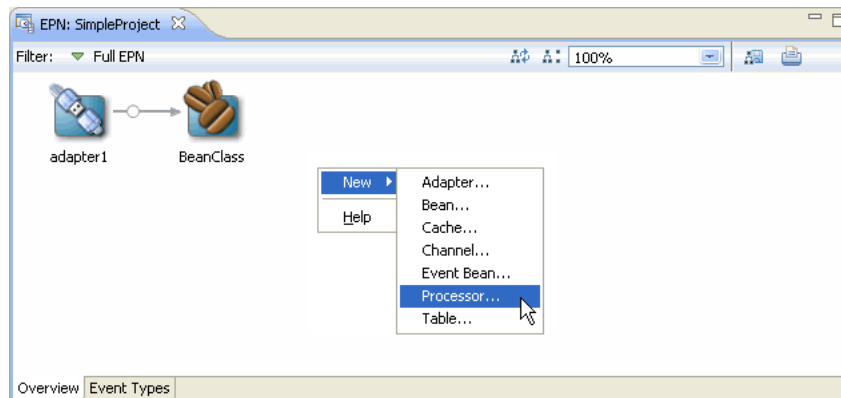
This section describes how to create a processor node using the EPN Editor. For information on creating other node types, see [Section , "How to Create a Basic Node"](#).

When deploying an Oracle Event Processing 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 Event Processing application configuration files. See [Section , "Overview of Component Configuration Files"](#) for more information about Oracle Event Processing configuration files.

#### To create a processor node:

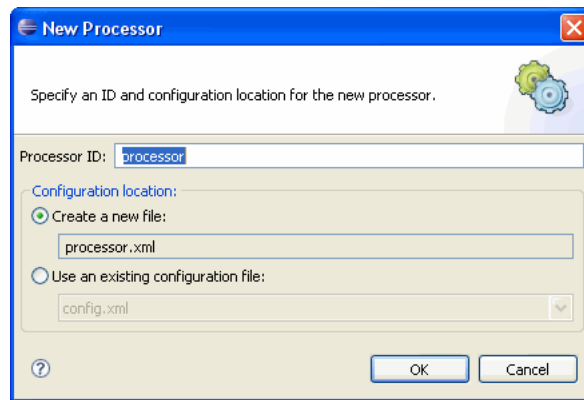
1. Open the EPN Editor (see [Section , "Opening the EPN Editor"](#)).
2. Right-click on an empty portion of the EPN Editor surface and select **New** from the context menu as [Figure 7–38](#) shows.

**Figure 7–38** *Creating a Processor Node*



3. Select node type **Processor**.

The New Processor dialog appears as shown in [Figure 7–39](#).

**Figure 7–39 New Processor Dialog**

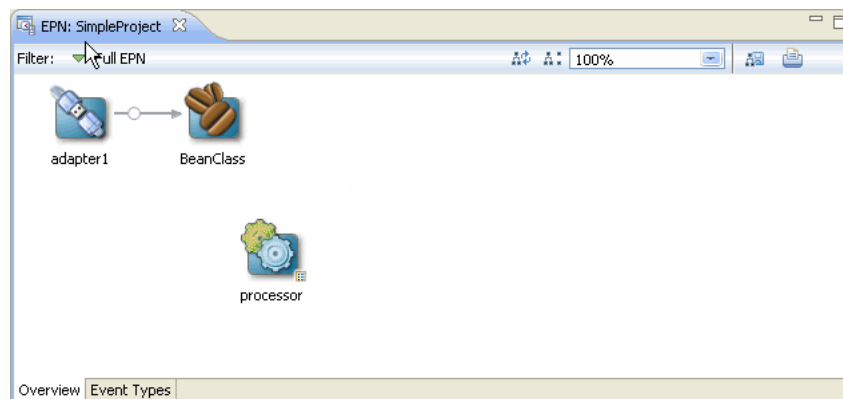
4. Configure the New Processor dialog as shown in [Table 7–4](#).

**Table 7–4 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 7–40](#) shows.

**Figure 7–40 New Processor Node**

To rename the node, see [Section , "Renaming Nodes"](#).

To reposition the node and update the EPN Editor layout, see [Section , "Laying Out Nodes"](#).

---



---

**Note:** In Oracle Event Processing, you must use a channel to connect a push event source to an Oracle CQL processor and to connect an Oracle CQL processor to an event sink. For more information, see [Section , "Channels Representing Streams and Relations"](#).

---



---

6. Optionally, configure additional processor options.

See:

- [Chapter 17, "Querying an Event Stream with Oracle CQL"](#)
- [Chapter 19, "Querying an Event Stream with Oracle EPL"](#)

## Connecting Nodes

The nodes in the EPN represent the flow of events through an Event Processing Network of an Oracle Event Processing 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.

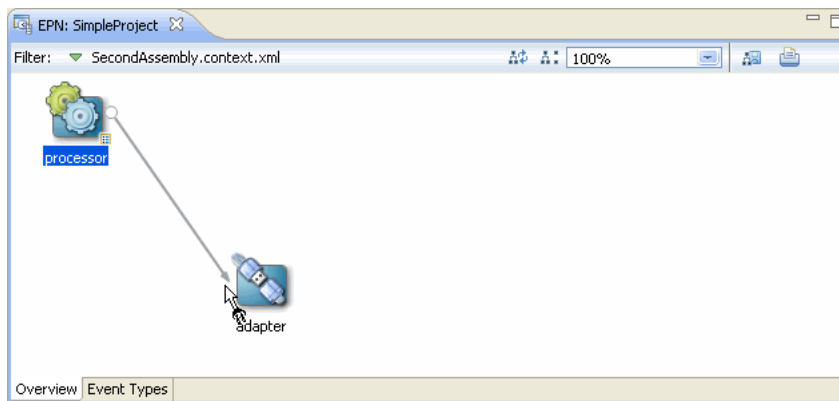
### How to Connect Nodes

This section describes how to connect nodes in the EPN Editor.

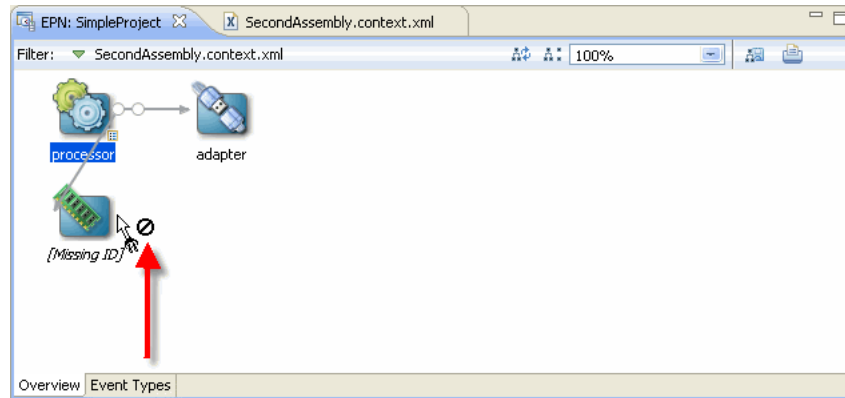
#### To connect nodes:

1. Open the EPN Editor (see [Section , "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 7-41](#) shows.

**Figure 7-41 Connecting Nodes: Connection Allowed**



- If the connection is not allowed, a forbidden icon is shown at the target end as [Figure 7-42](#) shows.

**Figure 7–42 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 , "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 7–43](#).

**Figure 7–43 Valid Connections**

[Figure 7–44](#) shows the EPN assembly file before connection.

**Figure 7–44 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 7–45](#) shows the EPN assembly file after connection.

**Figure 7–45 EPN Assembly File: After Connection**

```

<wlevs:adapter id="adapter" provider="httppub">
  <wlevs:listener ref="channel" />
</wlevs:adapter>

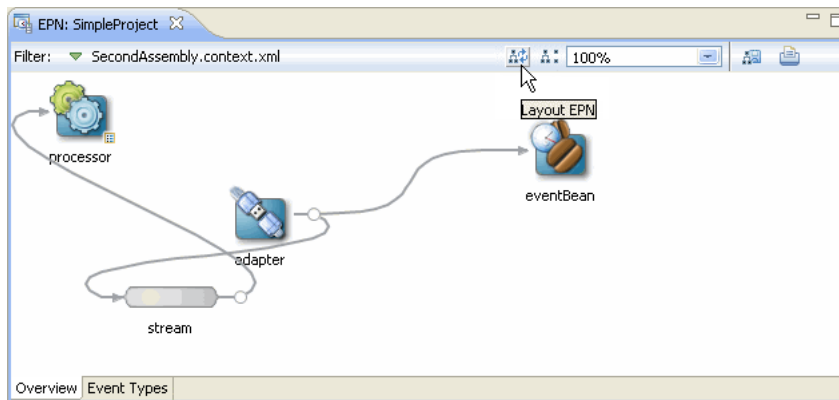
<wlevs:channel id="channel">
  <wlevs:listener ref="processor" />
</wlevs:channel>

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

```

## Laying Out Nodes

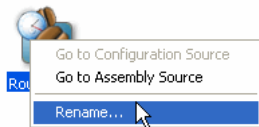
Certain EPN Editor actions will use the location of the action as the location of the rendered result. For example, when adding new nodes to an EPN using the EPN editor, a new node will appear at the location of the mouse click that was used to show the EPN Editor context menu. The user may not reposition 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 7–46](#) shows.

**Figure 7–46 Laying Out Nodes**

For more information, see [Section , "Layout"](#).

## 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 7–47](#) shows.

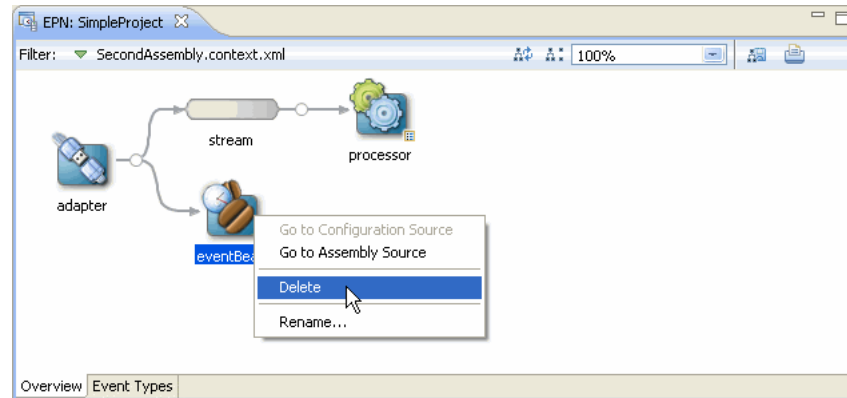
**Figure 7–47 Renaming Nodes**

## Deleting Nodes

You may delete most nodes and connections visible on the EPN Editor using the node's context menu or the **Delete** key:

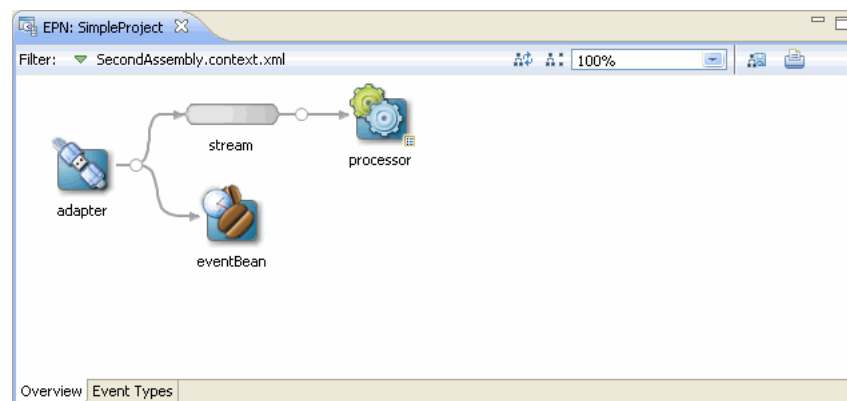
- Using the keyboard, select the object you want to delete and then click the **Delete** key.
- Using the context menu, right-click on the object to show the context menu, then select **Delete** as [Figure 7-48](#) shows.

**Figure 7-48 Deleting Nodes**



When deleting a node, the incoming and outgoing connections are also deleted. For example, [Figure 7-49](#) shows the EPN and [Figure 7-51](#) shows the assembly file before deleting the channel node named `stream`.

**Figure 7-49 EPN Before Deleting a Channel Node**



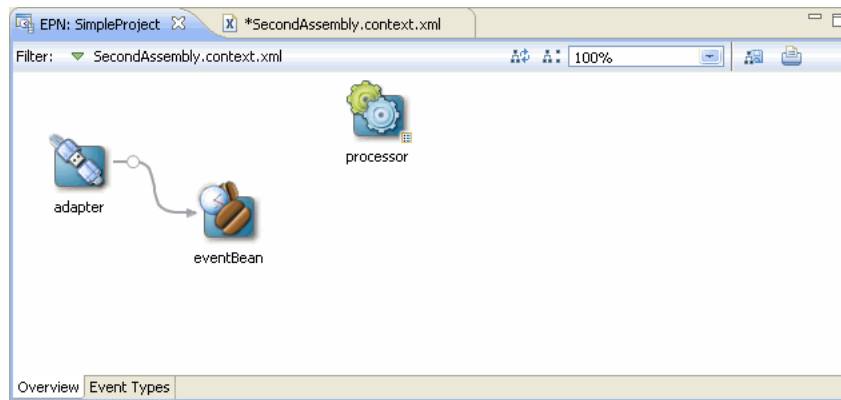
**Figure 7-50 Assembly File Before Deleting a Channel Node**

```
<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 7-51](#) shows the EPN and [Figure 7-52](#) shows the assembly file after deleting this channel node.

**Figure 7–51 EPN After Deleting a Channel Node****Figure 7–52 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

---

## Developing Oracle Event Processing Applications

Part III contains the following chapters:

- Chapter 8, "Walkthrough: Assembling a Simple Application"
- Chapter 9, "Defining and Using Event Types"
- Chapter 10, "Connecting EPN Stages Using Channels"
- Chapter 11, "Integrating the Java Message Service"
- Chapter 12, "Integrating an HTTP Publish-Subscribe Server"
- Chapter 13, "Integrating a Cache"
- Chapter 14, "Integrating Web Services"
- Chapter 15, "Integrating an External Component Using a Custom Adapter"
- Chapter 16, "Handling Events with Java"
- Chapter 17, "Querying an Event Stream with Oracle CQL"
- Chapter 18, "Configuring Applications With Data Cartridges"
- Chapter 19, "Querying an Event Stream with Oracle EPL"
- Chapter 20, "Configuring Event Record and Playback"
- Chapter 21, "Testing Applications With the Load Generator and csvgen Adapter"
- Chapter 22, "Testing Applications With the Event Inspector"



---

---

## Walkthrough: Assembling a Simple Application

This chapter introduces how to build Oracle Event Processing applications through a walkthrough in which you build a simple application. Along the way, it provides an overview of key concepts.

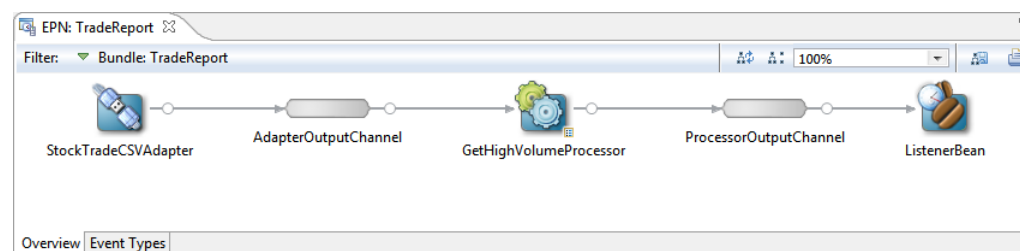
This chapter includes the following sections:

- [Introduction to the Simple Application Walkthrough](#)
- [Create the Workspace and Project](#)
- [Create an Event Type to Carry Event Data](#)
- [Add an Input Adapter to Receive Event Data](#)
- [Add a Channel to Convey Events](#)
- [Create a Listener to Receive and Report Events](#)
- [Set Up the Load Generator and Test](#)
- [Add an Oracle CQL Processor to Filter Events](#)
- [Summary: Simple Application Walkthrough](#)

### Introduction to the Simple Application Walkthrough

This walkthrough introduces the basics of building Oracle Event Processing applications using the Eclipse IDE. It is intended as a survey of key Oracle Event Processing concepts, a starting place from which you can investigate more about each.

The application you build in this walkthrough models a simple stock trade alert system. The application receives example data about stock trades, examines the data for certain characteristics, then prints some of the data to the console. The following illustration shows the application's finished EPN diagram:



This introduction includes the following sections:

- [Section , "Key Concepts in this Walkthrough"](#)

- [Section , "Before You Get Started"](#)

This walkthrough starts with [Section , "Create the Workspace and Project"](#).

## Key Concepts in this Walkthrough

This walkthrough introduces the following concepts that are typically part of Oracle Event Processing applications you build:

- IDE features designed to make building Oracle Event Processing applications easier, including a project type, a graphical editor for designing event processing networks, and validation support for project-specific configuration files
- Building an application as an event processing network (EPN), the core design construct for modeling the behavior of an application that receives streaming data and operates on that data as it flows through the application.
- Designing event types that model events, normalizing event data for use with code inside the application.
- Using adapters to manage interactions with external components, including sources of streaming data.
- Implementing a Java class that can receive or send events within an event processing network.
- Using Oracle Continuous Query Language (Oracle CQL) to filter events based on specific properties within them.

## Before You Get Started

You should have installed Oracle Event Processing and the Eclipse IDE. In addition, you should have updated the Eclipse IDE with the plugin included with Oracle Event Processing.

Although it introduces features specific to Oracle Event Processing, this walkthrough assumes that you are somewhat familiar with basic Java programming.

For more information, see the following topics:

- "Installation Overview" in the *Oracle Fusion Middleware Getting Started Guide for Oracle Event Processing*
- [Section , "Overview of Oracle Event Processing IDE for Eclipse"](#)

## Create the Workspace and Project

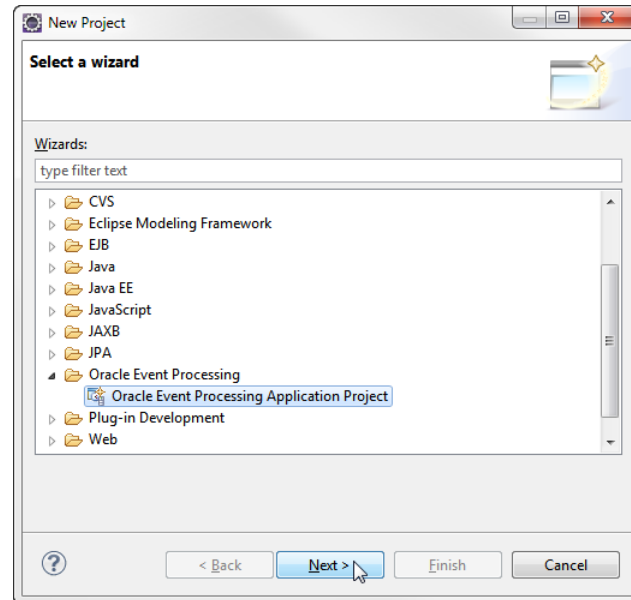
In this first step, you'll use the IDE to create the workspace and project in which to develop your application. To make it easier to develop Oracle Event Processing applications, the IDE provides the Oracle Event Processing application project type. This project type includes the artifacts and dependencies that typical Oracle Event Processing applications need, making it easier to get into writing application-specific code.

You should already have configured your IDE preferences to know the location of the JRockit JRE, which is the best practice choice on which to run the Oracle Event Processing server.

### Create the Walkthrough Workspace and TradeReport Project

1. Start the Eclipse IDE.

2. When prompted to select a workspace, create a workspace called `walkthroughs`.
3. From the **File** menu, choose **New > Project** to begin creating a new project.
4. In the **New Project** dialog, expand **Oracle Event Processing**, click **Oracle Event Processing Application Project**, then click **Next**.



5. In the **New Oracle Event Processing Application Project** dialog, in the **Project name** box, enter `TradeReport`.

Leave the **Use default location check** box selected to have the new project created in the location of the workspace you created.

6. Under **Target Runtime**, if no runtime is displayed in the dropdown, do the following:
  1. Click the **New** button.
  2. In the **New Server Runtime Environment** dialog, expand **Oracle**, then select **Oracle Event Processing v11.1** and click **Next**.
  3. Click **Next**.
  4. Under **New Oracle Event Processing v11.1 Runtime**, next to the **Oracle Middleware Home Directory** box, click **Browse**.
  5. In the **Browse for Folder** dialog, locate the Middleware home directory, then click **OK**.  
By default, the Middleware home directory will be located at `ORACLE_HOME/Middleware`.
  6. In the **JRE** dropdown, select a **JRockit JRE**.

The Oracle Event Processing server is optimized for use with the JRockit JRE. If a JRockit JRE isn't available among those listed, use the following steps to locate the JRE that is included with Oracle Event Processing.

Click the **Installed JRE preferences** link.

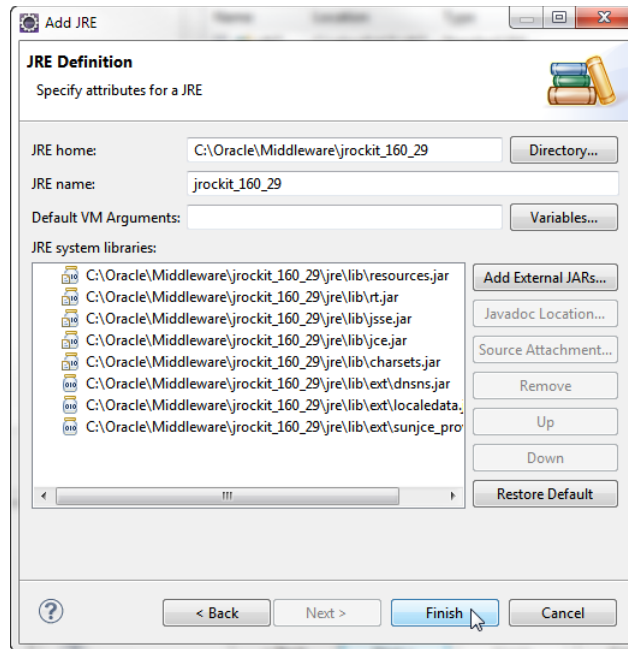
In the **Preferences** dialog, next to the **Installed JREs** list, click the **Add** button.

In the **Add JRE** dialog, under **Installed JRE Types**, select **Standard VM**, then click **Next**.

Next to the **JRE home** box, click the **Directory** button to browse for the location of the JRockit JRE.

In the **Browse For Folder** dialog, expand the Middleware home directory, then the **jrockit\_160\_29** directory, then click the JRE directory and click **OK**.

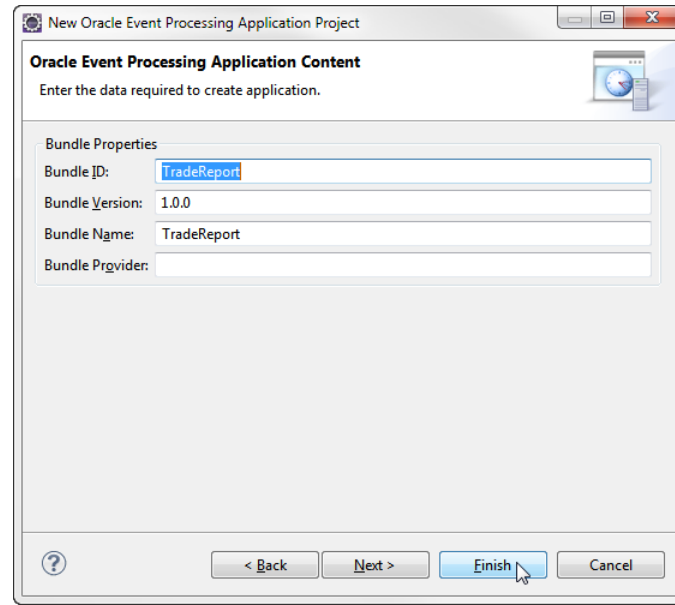
After the **Add JRE** dialog displays the JRE home directory, name, and system libraries, click the dialog's **Finish** button.



In the **Preference** dialog, select the check box for the JRE you just added, then click **OK**.

In the **New Server Runtime Environment** dialog, from the JRE dropdown, select the JRE you just added.

7. Click **Finish**.
7. In the **New Oracle Event Processing Application Project** dialog, with the **Oracle Event Processing v11.1** target runtime selected, click **Next**.
8. Under **Oracle Event Processing Application Content**, confirm the properties for the project you're created.



**9. Click Finish.**

After you've created your project, the IDE will display the Project Explorer, with a hierarchical list of the artifacts in your project, along with an empty editor for the event processing network you're about to build.

An event processing network (EPN) is a central design concept in an Oracle Event Processing application. An EPN represents the components, known as stages, that make up the application, as well as the path taken by events from stage to stage. As you develop your application, the EPN editor will display the stages you add and the connections between them. Conceptually, event data enters your application from the left, moving toward the right from stage to stage.

The design that the EPN editor shows is actually a graphical representation of the EPN's underlying configuration. When you add a stage or connection through the EPN editor, the IDE writes configuration XML to an underlying assembly file (just as editing the XML file directly will result in an updated graphical design in the EPN editor). An EPN assembly file includes the default configuration for each of an EPN's stages. This is a default configuration that cannot be changed on the server at runtime without redeploying the application. For configuration that can be edited at runtime, you can use another kind of configuration file, as described later.

---

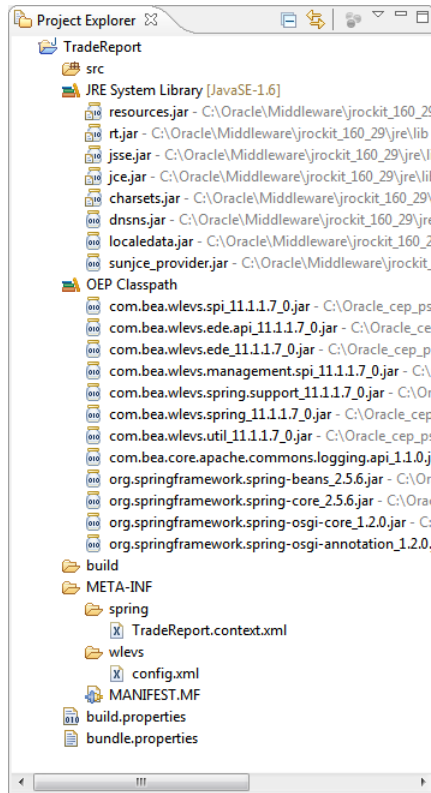
---

**Note:** The EPN assembly file's XML schema is an extension of the Spring framework configuration file. For more on Spring, see the Spring web site.

---

---

Take a look at the Project Explorer. You should have a project hierarchy that's something like what's shown in the following illustration.



If you're experienced with IDEs, much of this should be familiar. In addition to the usual places for Java source code, the JRE system library, and build output, you'll also find the following:

- **JAR files** that make up the Oracle Event Processing classpath. These provide the functionality needed for the Oracle Event Processing server, including for stages of an event processing network, Spring-configured components, and logging.
- A **spring directory** for configuration files that conform to the Spring framework (for more on Spring, see the Spring web site). The file you start off with is the EPN assembly XML file that describes the contents and structure of the event processing network you're building. As you build your EPN, adding and connecting stages, the IDE captures your work in this file. You can also edit the file manually. At this point, the file merely declares namespaces for the XML that will be added to it.
- A **wlevs directory** for files that describe components whose configuration should be editable at runtime. Most components have a default configuration that you can override within a component configuration file in the wlevs directory. The only component whose configuration you *must* put in one of these files is a processor. A processor's Oracle CQL code is editable at runtime by using the Oracle Event Processing Visualizer.

If you're configuring multiple components, you can use one or multiple component configuration files. For example, in the case of a team of developers, where each is responsible for a different component, you might want to have a configuration file per component to avoid source control conflicts.

Now that you've got a project created, in [Section , "Create an Event Type to Carry Event Data"](#), you'll start assembling an event processing network. You'll begin by



creating an event type that will represent incoming event data to the application's logic.

## Create an Event Type to Carry Event Data

Oracle Event Processing applications are about receiving, processing, and sending events. Events start as event data that can be in almost any structured form. The event data arrives at the application in its raw form, then is bound to an event type that you define. Using your own event type makes handling the data predictable for the rest of your code in the application, including Oracle CQL queries, Java code, and so on. (The conversion is done by an adapter. More about those later.)

So an early task in defining any event processing network is clarifying the structure of the data coming from the source, then defining the form to which you'll convert that data for use inside the EPN. In this section, you'll work from the structure of example stock trade data to define an event type to which the data will be bound.

In the case of the application you're building here, the sample incoming event data is arriving as rows of comma-separated values. Each row contains data about a particular thing that happened -- here, a stock trade. The structure of the values is consistent from row to row. Here are a few example rows:

```
IBM,15.5,3.333333333,3000,15
SUN,10.8,-1.818181818,5000,11
ORCL,14.1,0.714285714,6000,14
GOOG,30,-6.25,4000,32
YHOO,7.8,-2.5,1000,8
```

Though they aren't labeled in the CSV file, the values could be labeled as follows:

```
stock symbol, price, percentage change, volume of shares purchased, last price
```

That's the structure of the incoming trade event data. In order to handle that data in your application -- query it, filter it, perform calculations, and so on -- you will need to assign that data to a new structure that supports doing those things. In particular, the new structure should have the ability to specify properties of particular types. Having types matched to the anticipated values makes it easier to handle the values in code.

Looking at the example data and labels, you can imagine each of the values for each row bound to properties of the following Java types:

```
String, Double, Double, Integer, Double
```

Oracle Event Processing supports several forms on which you can base your new event type. These include JavaBean classes, tuples, and `java.util.Map` instances. A JavaBean class is the best practice type for new event types, so that's what you will use here for trade events.

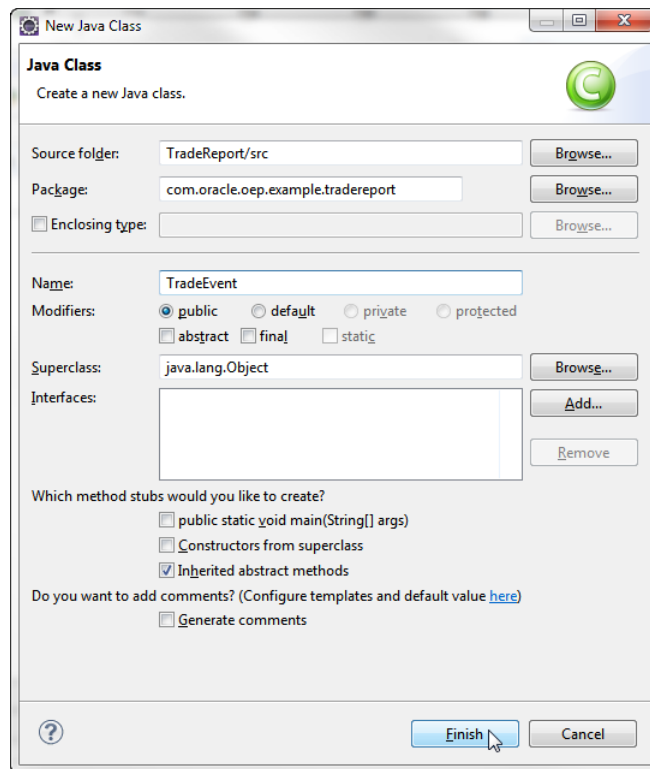
If you haven't worked with them before, you should know that JavaBeans are Java classes that follow specific standard rules designed to make them predictable. In general, the idea is for the class to provide variables to hold data inside it (such as the trade event stock symbol) and methods through which the data can be retrieved or set by code outside the class. Naming the methods with predictable names, such as "setSymbol" and "getSymbol", gives code outside the class a predictable way to use the data inside the class. For example, when Oracle Event Processing needs to bind incoming event data to your event type, it will create a new instance of the event type JavaBean and use the bean's "set" methods to set the data in the proper places inside the class. Code later in the application, such as Java code or Oracle CQL code you will write, will be able to call the bean's "get" methods to get the data out again.

In the following sections, you will create the JavaBean as code behind your event type, then configure the event type to use the JavaBean.

### Create the TradeEvent JavaBean

In this section, you will create a TradeEvent JavaBean class that will be used as an event type for incoming trade event data.

1. In the IDE, right-click the src directory, then click **New > Class**.
2. In the **New Java Class** dialog, in the package box, enter `com.oracle.oep.example.tradereport`
3. In the **Name** box, enter `TradeEvent`.



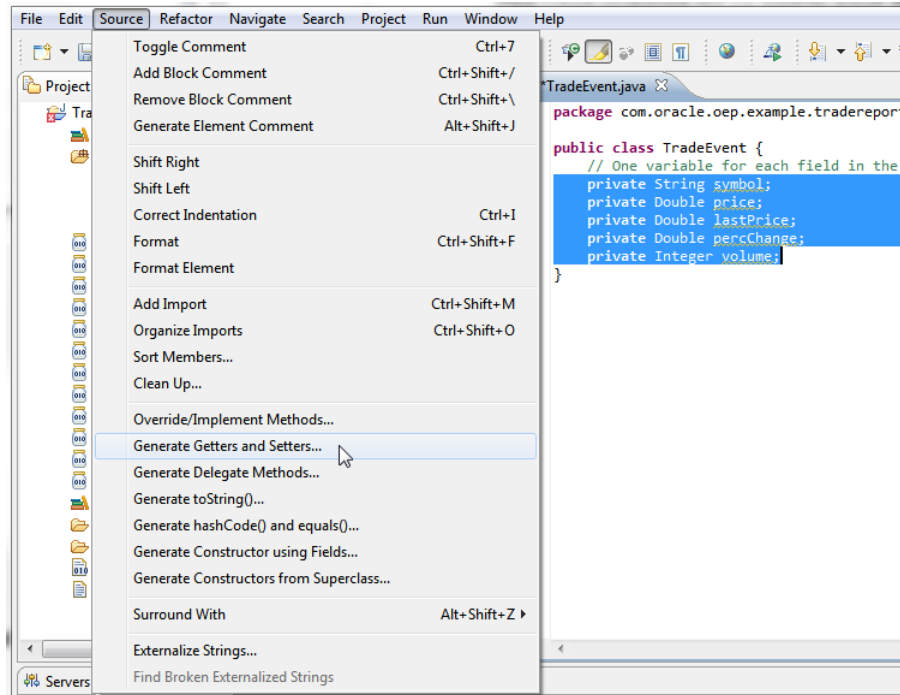
4. Click **Finish**.

The TradeEvent.java source code window will display with the declaration for the TradeEvent class.

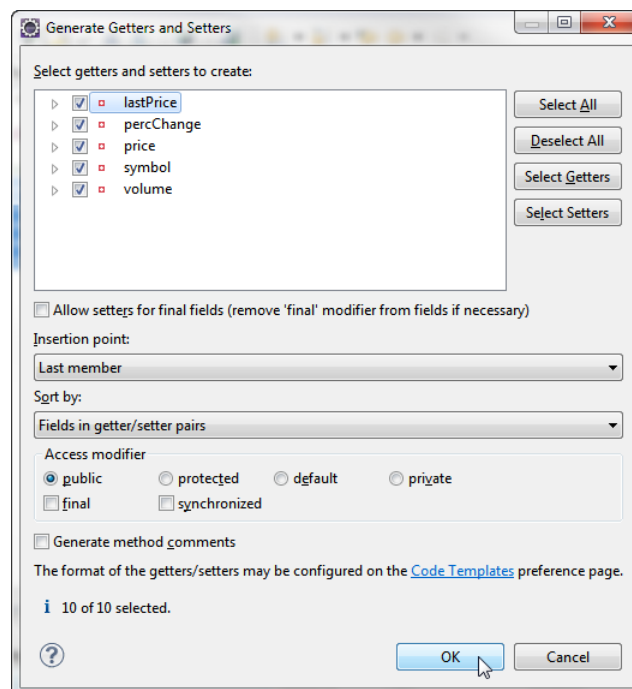
5. In the TradeEvent class, just beneath the TradeEvent class declaration, add private variables for each of the properties you'll need as shown in the following example.

```
public class TradeEvent {
    // One variable for each field in the event data.
    private String symbol;
    private Double price;
    private Double lastPrice;
    private Double percChange;
    private Integer volume;
}
```

6. Select the code for all of the variables, click the **Source** menu, then click **Generate Getters and Setters**.



7. In the **Generate Getters and Setters** dialog, select the check box for each variable you selected (in other words, all of them).
8. In the **Insertion Point** dropdown, select **Last member**.



9. Click **OK**.

In the TradeEvent source code window, notice that you have created pairs of methods that simply get or set the values of the variables you added.

10. Save the TradeEvent.java file and close the code window.

And that's all there is to creating a JavaBean (at least, in this case). The next step is to tell your application that the JavaBean should be used as an event type.

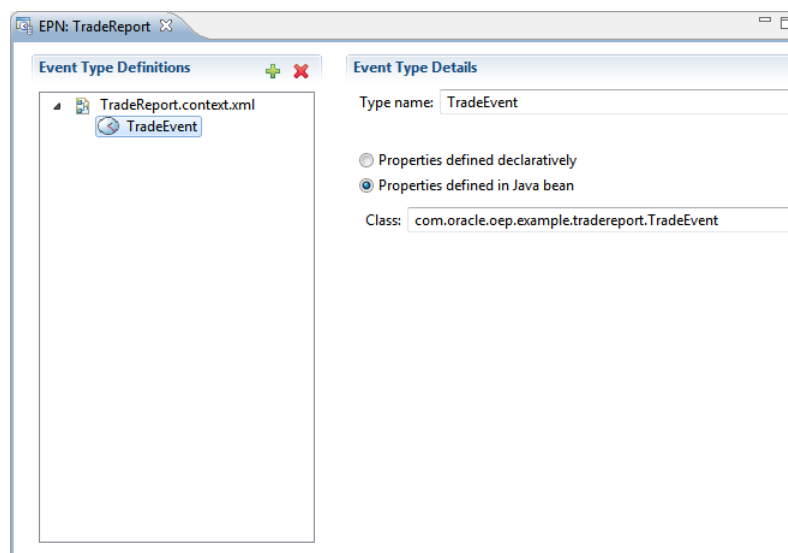
### Configure the TradeEvent Event Type

In this section, you'll tell the TradeReport application that the TradeEvent JavaBean should be used as an event type.

1. In the IDE, confirm that the empty TradeReport EPN editor is open. Its tab at the top of the window should read **EPN: TradeReport**.
2. At the bottom of the designer window, click the **Event Types** tab.
3. Under **Event Type Definitions**, select **TradeReport.context.xml**. This is the EPN assembly file underlying your EPN design.

This is also the file you noticed in the META-INF/spring directory of the project explorer. You are selecting it here to ensure that the event type you are about to configure gets defined in that file.

4. Under **Event Type Definitions**, click the plus sign to add a new event definition.
5. Click the **newEvent** entry that was created.
6. Under **Event Type Details**, in the **Type** name box, change the event type's name to TradeEvent. (It doesn't have to be the same as the JavaBean class, but it makes things simpler to use the same name.)
7. Select the **Properties defined in Java bean** option. The other option is for defining events as tuples.
8. In the **Class** box, enter the name of the JavaBean class you created: `com.oracle.oep.example.tradereport.TradeEvent`.



9. Save the file to have the event type name updated in the event type definitions.

If you're curious about what was added to your EPN assembly file, open the file. It's in the Project Explorer under TradeReport > META-INF > spring >

TradeReport.context.xml. The part you added looks something like the following (you might need to click the Source tab at the bottom of the editor window):

```
<wlevs:event-type-repository>
  <wlevs:event-type type-name="TradeEvent">
    <wlevs:class>com.oracle.oep.example.tradereport.TradeEvent</wlevs:class>
  </wlevs:event-type>
</wlevs:event-type-repository>
```

As the code suggests, Oracle Event Processing manages event types in an event type repository. The TradeEvent event type you defined is mapped to the TradeEvent JavaBean class you created.

Now that you have an event type defined and configured, you need to tell the application how to watch for incoming event data. You also need to specify that the data should be assigned to the TradeEvent type you created. You do those things with an adapter, which you will add in [Section , "Add an Input Adapter to Receive Event Data"](#).

## Add an Input Adapter to Receive Event Data

Before you can try out the event processing network you're building, you will need to create a way for event data to flow into it. In Oracle Event Processing applications, adapters manage traffic between external components and the internals of the event processing network. Event data arriving from external sources enter the application through an adapter, while event data leaving the application on its way to external components or applications exits through an adapter. By default, Oracle Event Processing includes adapters for three different kinds of external components: Java Message Service (JMS) destinations, HTTP publish-subscribe servers, and CSV files.

The adapter for CSV files, called csvgen, is great for trying things out with code in development. It's a lightweight alternative to the task of defining an adapter for a more substantial, "real" event data source. In your own apps, you might find it easier to use the csvgen adapter until you have your EPN started and some of its logic defined.

The csvgen adapter's logic knows how to translate event data read from a CSV file into the event type you defined. You use the csvgen adapter in conjunction with the load generator utility included with Oracle Event Processing (more about the load generator later in this walkthrough). The csvgen adapter's implementation code is included by default in Oracle Event Processing, so all you need to do is add configuration code to declare the adapter as a stage in your EPN, as well as set a few of its properties.

### Add Code to Configure a CSV Adapter

In this section, you will add configuration code that creates a place for your CSV adapter in the event processing network. The code will also set a few adapter properties needed to configure the adapter's runtime behavior. You add the code to the EPN assembly file so that the adapter will be included in the EPN and appear in the EPN designer as a stage.

1. In the IDE, locate and open the EPN assembly file. In the Project Explorer, it should be located at **TradeReport > META-INF > spring > TradeReport.context.xml**.
2. Below the event-type-repository XML stanza, add the following XML to declare the adapter:

```
<wlevs:adapter id="StockTradeCSVAdapter" provider="csvgen">
```

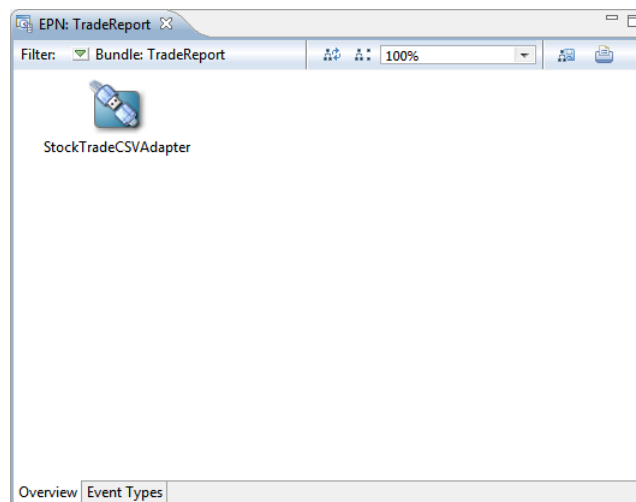
```

<wlevs:instance-property name="port" value="9200" />
<wlevs:instance-property name="eventName"
  value="TradeEvent" />
<wlevs:instance-property name="eventPropertyNames"
  value="symbol,price,percChange,volume,lastPrice" />
</wlevs:adapter>

```

This XML stanza declares an instance of the csvgen adapter and assigns to it three properties that configure it for use in your EPN. The adapter uses the properties to map from incoming raw event data to the properties of the event type you defined in [Section , "Create an Event Type to Carry Event Data"](#). The following describes the values you're adding:

- **Adapter declaration.** The `id` attribute value is a unique identifier for the adapter. The provider attribute value must be "csvgen" in order to refer to the csvgen implementation included with Oracle Event Processing.
  - The `port` instance property tells the adapter instance what port to listen on for incoming event data. The value here, 9200, corresponds to the port number to which the load generator will send event data (more on that later).
  - The `eventName` instance property tells the instance the name of the event type to which incoming event data should be assigned. Here, you give the name of the TradeEvent type you defined earlier.
  - The `eventPropertyNames` instance property tells the instance the names of the event type properties to which data should be assigned. Notice in this case that the `eventPropertyNames` attribute value is a comma-separated list of the same properties you defined in the JavaBean for the event type. In order for the csvgen adapter to map from incoming values to event type properties, the names here must be the same as your event type and must be in the same order as corresponding values for each row of the CSV file.
3. Save and close the EPN assembly file with the adapter XML in it.
  4. If it isn't open already, open the EPN editor to its **Overview** tab.
  5. In the EPN editor, notice that it now displays an icon representing the csvgen adapter instance you just added. The icon will be labeled with the adapter id value you specified in configuration code.



In the next step, [Section , "Add a Channel to Convey Events"](#), you will add a way to carry events from the adapter to logic you will add in a moment.

## Add a Channel to Convey Events

In this step, you will add a way to connect the adapter you added in [Section , "Add an Input Adapter to Receive Event Data"](#) to a bit of logic you will add in the next step. Afterward, you will test the application.

In Oracle Event Processing applications, you connect EPN stages together by using a channel. A channel is a conduit that transfers events from one part of the EPN to another. Though conveying events is a channel's primary purpose, channel configuration options give you opportunities to specify other properties through which you can tune the application. These include:

- Whether the channel can process events asynchronously, and how big the buffer for this can get.
- How many threads may be used to process events in the channel (a larger number can increase performance).
- Whether to partition events, based on their properties, in order to have the events dispatched to separate downstream parts of the EPN.

Those kinds of channel configuration properties have default values, so you needn't set them for the application you're building. In this application, you'll keep channel configuration simple.

The purpose of the channel you are about to add is to carry newly generated events from the adapter that's receiving event data to code you will add in the next step.

### Add the AdapterOutputChannel

1. In the IDE, ensure that the EPN editor is open and that the EPN you're building is visible.
2. Right-click the EPN editor, then click **New > Channel**.  
A channel icon should appear in the EPN editor.
3. Double-click the channel icon that appears in the editor to display the channel's configuration XML in the EPN assembly file.
4. In the assembly XML file, locate the channel XML stanza for the channel you added. It should look something like the following:

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

You might notice a warning that "channels should declare an 'event-type' value". You are about to fix that.

5. Edit the default XML to match the following (or simply paste the following code over the code you have).

```
<wlevs:channel id="AdapterOutputChannel" event-type="TradeEvent">
</wlevs:channel>
```

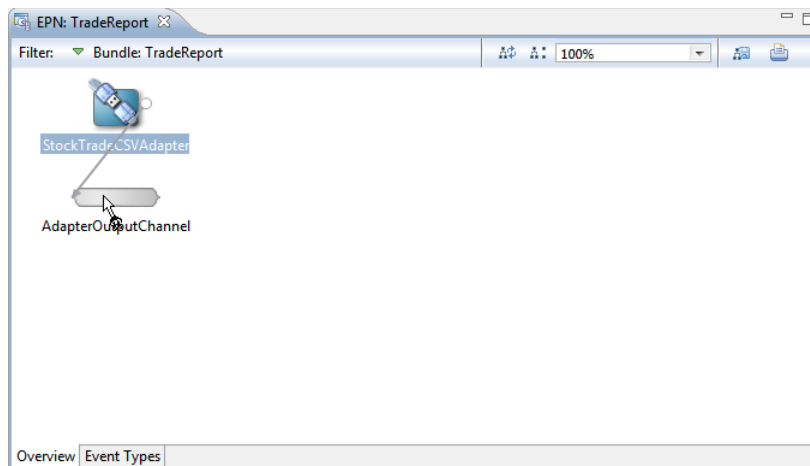
The following describes the values you're adding to configure the channel:

- The `channel` element represents configuration for the channel, placing the channel in the EPN.

- The `id` attribute's value is a unique identifier for the channel in this EPN. (You will be adding another channel.)
- The `event-type` attribute's value is the name of the event type that the channel is configured to convey. In this case, you're setting it to `TradeEvent`, the name of the event type you added earlier, whose implementation is the `TradeEvent` JavaBean you created. Note that the value you are adding here is the same as the `type-name` attribute value of the `event-type` element elsewhere in the assembly file.

After you have finished editing the channel configuration XML, save the assembly file.

6. Return to the EPN editor. You might need to click the **EPN: TradeReport** tab to display the EPN editor.
7. In the EPN editor, create a connection from the input adapter to the channel you added. To do this, click the **StockTradeCSVAdapter** icon and drag to the **AdapterOutputChannel** icon. This will create a connecting line between the two icons.



8. To make the EPN diagram tidy by having icons display sequentially from left to right, click the **Layout EPN** icon at the top right corner of the EPN editor.
9. Double-click the **StockTradeCSVAdapter** icon to display the adapter's configuration XML in the assembly file.
10. In the assembly file, notice that creating a connection between the adapter and the channel has added a `listener` element to the adapter XML stanza. That element's `ref` attribute value is set to the `id` attribute of channel element. This XML defines the connection that is graphically displayed in the EPN editor.

The resulting EPN assembly XML should look something like the following (aspects that connect components are shown in bold text):

```
<?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:jdbc="http://www.oracle.com/ns/ocep/jdbc"
  xmlns:spatial="http://www.oracle.com/ns/ocep/spatial"
  xsi:schemaLocation="...">
  <!-- Schema locations omitted for brevity. -->
```



```

        <wlevs:event-type-repository>
            <wlevs:event-type type-name="TradeEvent">
                <wlevs:class>com.oracle.cep.example.tradereport.TradeEvent</wlevs:c
lass>
            </wlevs:event-type>
        </wlevs:event-type-repository>

        <wlevs:adapter id="StockTradeCSVAdapter" provider="csvgen">
            <wlevs:listener ref="AdapterOutputChannel" />
            <wlevs:instance-property name="port" value="9200" />
            <wlevs:instance-property name="eventName"
                value="TradeEvent" />
            <wlevs:instance-property name="eventPropertyNames"
                value="symbol,price,percChange,volume,lastPrice" />
        </wlevs:adapter>

        <wlevs:channel id="AdapterOutputChannel" event-type="TradeEvent">
        </wlevs:channel>

    </beans>

```

11. Save TradeReport.context.xml and close the file.

In this step, you added a channel to convey events out of the input adapter. In the next step, [Section , "Create a Listener to Receive and Report Events"](#), you will add a stage for the events to go.

## Create a Listener to Receive and Report Events

In this step, you will add a stage that will complete a simple event processing network so that you can test the application with event data. The stage will be a "listener" Java class that's designed to receive trade events passing through the EPN and report information about the trades.

The listener you're adding is a particular kind of Java class known in Oracle Event Processing as an event sink. An *event sink* is code that is able to receive events as they pass through an EPN. By intercepting events with an event sink, you can use the class's logic to find out what's inside the events and use that data while executing other logic in the class. In addition to writing event sinks, you can also write event sources, which are able to send events to stages downstream in the EPN.

So you can imagine, for example, a single Java class that intercepts events, executes code that changes the events' contents (or creates new events from them), then sends the resulting events along to the next stage. Such code could also initiate other processes (potentially in other applications) based on what the events look like.

Both event sinks and event sources implement particular Oracle Event Processing Java interfaces. For example, an event sink implements one of two interfaces, depending on whether the events it's receiving are part of a stream or a relation:

```
com.bea.wlevs.ede.api.StreamSink or com.bea.wlevs.ede.api.RelationSink.
```

The difference between streams and relations isn't especially important in the application you are building here (your code won't rely on the difference). Still, it's worth a brief explanation because it's an important part of understanding how event processing with Oracle Event Processing works.

A stream is a sequence of events that's sequential with respect to time, with events with an earlier timestamp arriving before those with a later timestamp. In contrast, a relation is a set of events in which events in the set were chosen because they met

certain criteria. Code executing on a relation can result in delete, insert, and update events.

As an illustration, consider a metaphor for a stock trade application that's more involved than the one you're building here. Imagine that the trade events your application receives are flowing through a pipe that you can't see into. The events emerge from the pipe in the order they were received -- an order that you care about because it matters which events occurred before which other events. To catch them, you put a bucket under the end of the pipe that's designed to catch fifteen seconds' worth of the trades at a time so that your code can see if a trade of one specific stock occurred within 15 seconds before a trade of another specific stock.

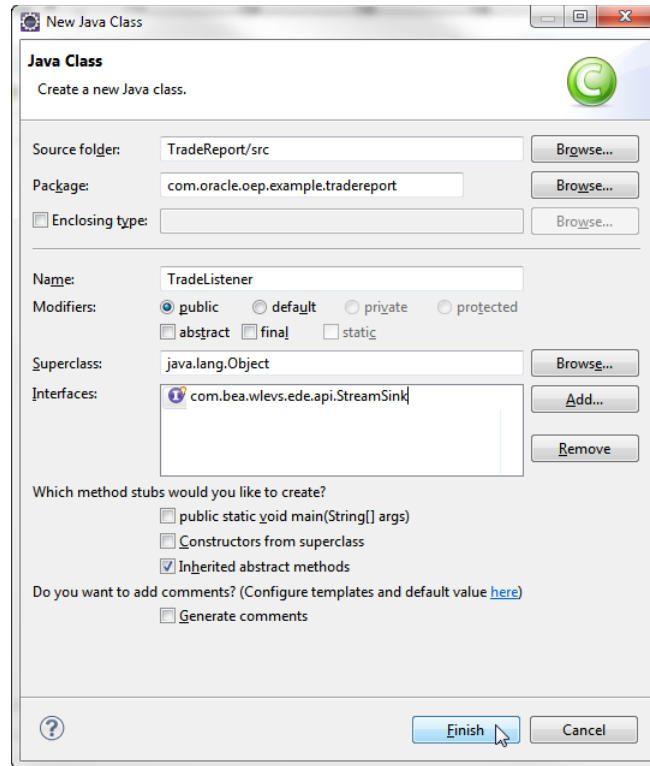
The pipe is a stream, where the sequence and adjacent quality are key characteristics. The bucket is a relation, where certain shared characteristics define what's in the bucket.

In the following sections, you'll create a Java class that implements the interface through which it listens for events, then configure that class as an event bean, making it part of the EPN.

### Create the Listener Event Sink Class

In this section, you'll create a Java class capable of receiving events from a stream.

1. In the IDE, in the **Project Explorer**, right-click the **src** directory, then click **New > Class**.
2. In the **New Java Class** dialog, in the **Package** box, enter `com.oracle.oep.example.tradereport`.
3. In the **Name** box, enter `TradeListener`.
4. Next to the **Interfaces** box, click the **Add** button to select the interface your listener will need to implement to be an event sink.
5. In the **Implemented Interfaces Selection** dialog, in the **Choose interfaces** box, enter `com.bea.wlevs.ede.api.StreamSink`.
6. With **StreamSink** selected in the **Matching items** box, click **OK**.
7. In the **New Java Class** dialog, under **Which method stubs would you like to create**, ensure that the **Inherited abstract methods** check box is selected (the others should be cleared).



**8. Click Finish.**

The `TradeListener.java` source code window will open to display the declaration for the `TradeListener` class. A declaration for the `onInsertEvent` method should also be present in the `.java` file. This method is required when implementing the `StreamSink` interface.

**9. In the `TradeListener` class, edit the `onInsertEvent` method to match the following code:**

```
@Override
public void onInsertEvent(Object event) throws EventRejectedException {

    if (event instanceof TradeEvent){
        String symbolProp = ((TradeEvent) event).getSymbol();
        Integer volumeProp = ((TradeEvent) event).getVolume();
        System.out.println(symbolProp + ":" + volumeProp);
    }
}
```

This is the method that Oracle Event Processing will use to pass an event into your listener, where your code can do something with it. The changes you're making implement the method to do the following:

- Ensure that the incoming event is an instance of your `TradeEvent` `JavaBean`. (Any others would be ignored.)
- If the event is a `TradeEvent`, the code will simply print out the stock symbol and trade volume it contains.

**10. Save and close the `TradeListener.java` file.**

### Configure the Listener Class as an Event Bean

In the preceding section, you implemented an event sink to create a class to listen for events of a particular type. In this section, you will add that listener to your EPN as an event bean that you can connect to other parts of the EPN.

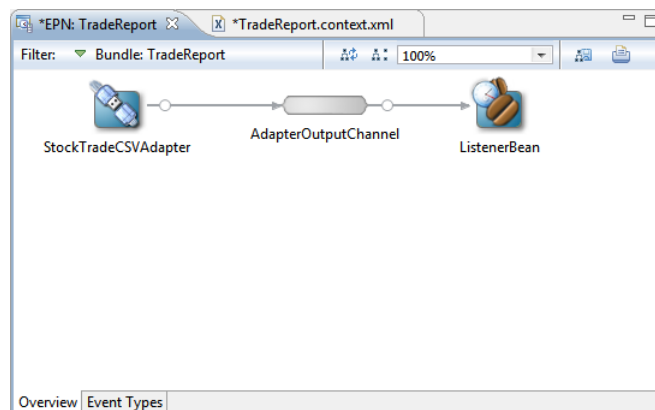
An event bean is a way to add Java code to an EPN. Another way is a Spring bean. Though they offer differing management features, both bean models provide a means to configure an EPN stage whose implementation is Java code that you write. An event bean, which you're creating here, integrates more fully with Oracle Event Processing server features. A Spring bean is a good choice when you need to integrate with an existing Spring framework.

1. In the EPN editor, right-click an empty area, then click **New > Event Bean**.
2. Double-click the new **eventBean** icon to display its configuration code in the assemble XML file.
3. In the `TradeReport.context.xml` source code, locate the `event-bean` element and edit it so that it appears as follows:

```
<wlevs:event-bean id="ListenerBean"
  class="com.oracle.oep.example.tradereport.TradeListener" />
```

With this code, you're configuring the event bean with the unique identifier, "ListenerBean", and an implementation class that's the class you created earlier.

4. Save the assembly XML file.
5. Return to the EPN editor to see that the design reflects your changes to the underlying XML, such as the new event bean name.
6. In the EPN editor, click the **AdapterOutputChannel** to select it.
7. Again click the **AdapterOutputChannel** and drag to the **ListenerBean** event bean icon. This creates a connection so that events can pass from the channel to the event bean.
8. Tidy the diagram by clicking the **Layout EPN** icon.



9. Double-click the **AdapterOutputChannel** to display its underlying XML.

Note that the channel XML stanza now includes a listener element whose `ref` attribute value is the same as the ID value for the event bean you added. (The word "listener" in both places here is for convenience only.)

```
<wlevs:channel id="AdapterOutputChannel" event-type="TradeEvent">
  <wlevs:listener ref="ListenerBean" />
```

```
</wlevs:channel>

<wlevs:event-bean id="ListenerBean"
  class="com.oracle.oep.example.tradereport.TradeListener" />
```

#### 10. Save TradeReport.context.xml.

In this step, you added an event bean to listen for events traveling through the event processing network. In the next step, [Section , "Set Up the Load Generator and Test"](#), you will test the application you are building.

## Set Up the Load Generator and Test

In this step, you will deploy your project on an instance of the Oracle Event Processing server, then use the load generator utility included with Oracle Event Processing to feed sample data to the application you are building.

The load generator utility installed with Oracle Event Processing provides a way for you to easily begin testing your project. It's specifically designed to read a CSV file and send rows in the file as event data to a port you specify in its configuration. In your project, a CSV adapter listening on that port receives the rows and converts their data into instances of the event type you defined for it. Through a properties file, you can configure certain aspects of the load generator, including the CSV file and target port to use, how long the load generator should run, how fast it sends event data, and so on.

In the following sections, you'll create an instance of the Oracle Event Processing server, configure it for deploying your project, then set up the load generator and test the project.

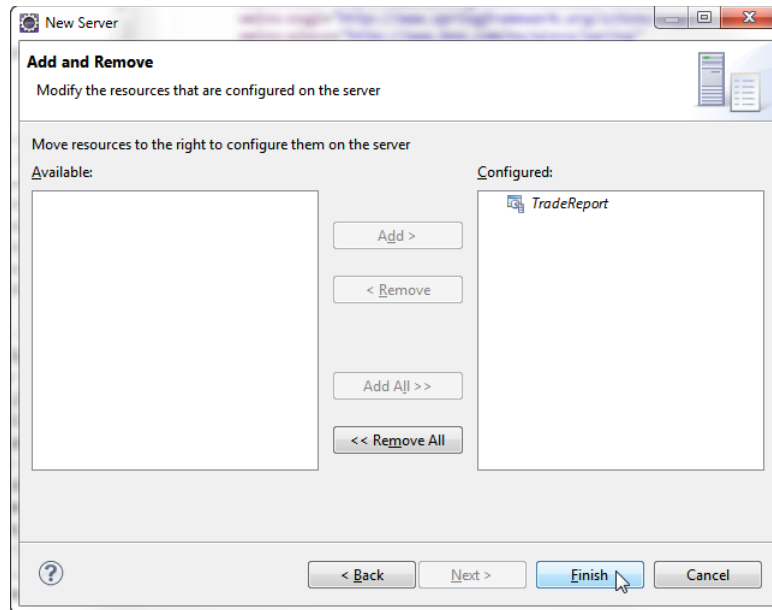
### Create a Server on Which to Run the Project

In this section, you'll add an Oracle Event Processing server instance to the TradeReport project. Adding the server instance gives you a convenient way to use the IDE to start and stop the server, as well as redeploy the project whenever you make changes and want to test again.

The IDE is aware of the Oracle Event Processing server through the server runtime environment included when you created the project. (You might have needed to add the server runtime environment as a separate step.)

1. In the IDE, ensure that the **Servers** view is displayed. If it isn't visible, click **Window > Show View > Servers** to display it.
2. In the **Servers** view, right-click to display the context menu, click **New**, then click **Server**.
3. In the **New Server** dialog, under **Select the server type**, expand **Oracle**, then click **Oracle Event Processing v11.1**.
4. Leave the host name, server name, and runtime environment as is, then click **Next**.
5. Under **New Oracle Event Processing v11.1 Server**, ensure that the **Local Server** option is selected, then click **Next**.
6. In the next window, under **New Oracle Event Processing v11.1 Server**, leave the **Domain Directory** as is, then click **Next**.
7. Under **Add and Remove**, in the **Available** box, select **TradeReport**, then click the **Add** button between the **Available** and **Configured** boxes.

Moving the TradeReport project name into the Configured list specifies that the TradeReport project you're building should be configured to run on the server you're adding. This simple step makes it easy to deploy (and redeploy) the project to the server as you're debugging.



**8. Click Finish.**

After you've added the server, it should be listed in the Servers view as Oracle Event Processing v11.1 at localhost (Stopped). Expand its entry to confirm that the TradeReport project is one of its configured projects.

### Set Up the Test Data and Load Generator

You don't actually need to set up the test data and load generator -- they're set up when you install Oracle Event Processing. But since you'll be using them to debug your project, use the following steps to take a look at them.

1. In the text editor of your choice, open the StockData.csv file included with Oracle Event Processing. By default, you'll find this file at the following path:

```
ORACLE_HOME/Middleware/ocep_11.1/utills/load-generator/StockData.csv
```

Notice that the contents of the file are essentially more of what you saw in the example when you defined an event type for the data. Notice that, as with any CSV file, the rows are uniform in terms of the order of values they contain.

2. In the text editor, open the StockData.prop file.

This is the properties file that configures the work of the load generator. Two of its properties -- `test.csvDataFile` and `test.port` -- are required in order for the load generator to work. The other properties are technically optional, but you'll need to set one more in order to ensure that the load generator knows that your input is in CSV form. To debug the TradeReport project, you should have the following properties set:

- `test.csvDataFile` -- The name of the CSV file that the load generator will read. The value here should be `StockData.csv`.

- `test.port` -- The port number to which the load generator will send event data. This should be the port value you specified when you configured the CSV adapter instance, or 9200.
- `test.packetType` -- The form that the load generator will be handling. This value should be CSV.

### Debug the Project

Of course, you probably won't need to *actually* debug this project (you've been doing everything exactly as described here, right?). But setting a breakpoint and running the Oracle Event Processing server in debug mode will give you a chance to see how things are working.

1. In the IDE, open the **TradeListener.java** file you created in an earlier step. In the **Project Explorer**, the file should be visible by expanding **TradeReport > src > com.oracle.cep.example.tradereport**.
2. In **TradeListener.java**, set a breakpoint at the line with the following code:
 

```
System.out.println(symbolProp + ":" + volumeProp);
```
3. In the **Servers** view, select the server you added: **Oracle Event Processing v11.1 at localhost**
4. With the server selected, click the **Start the server in debug mode** button in the upper right corner of the **Servers** view.

You will likely need to wait for a few moments while the server starts, the IDE compiles your project, and then deploys the **TradeReport** project as an application to the running server. During this time, the Console view will display status messages related to the server's startup progress. The server is running in debug mode when the Console view's output ends with `<Server STARTED>` or `<The application context for "TradeReport" was started successfully>`.

5. In the **Servers** view, confirm that the project does not need to be republished to the server. If the **TradeReport** entry shows **(Republish)**, then right-click the **TradeReport** entry and click **Force Publish**. After it has been successfully deployed, the entry will show **(Synchronized)**.

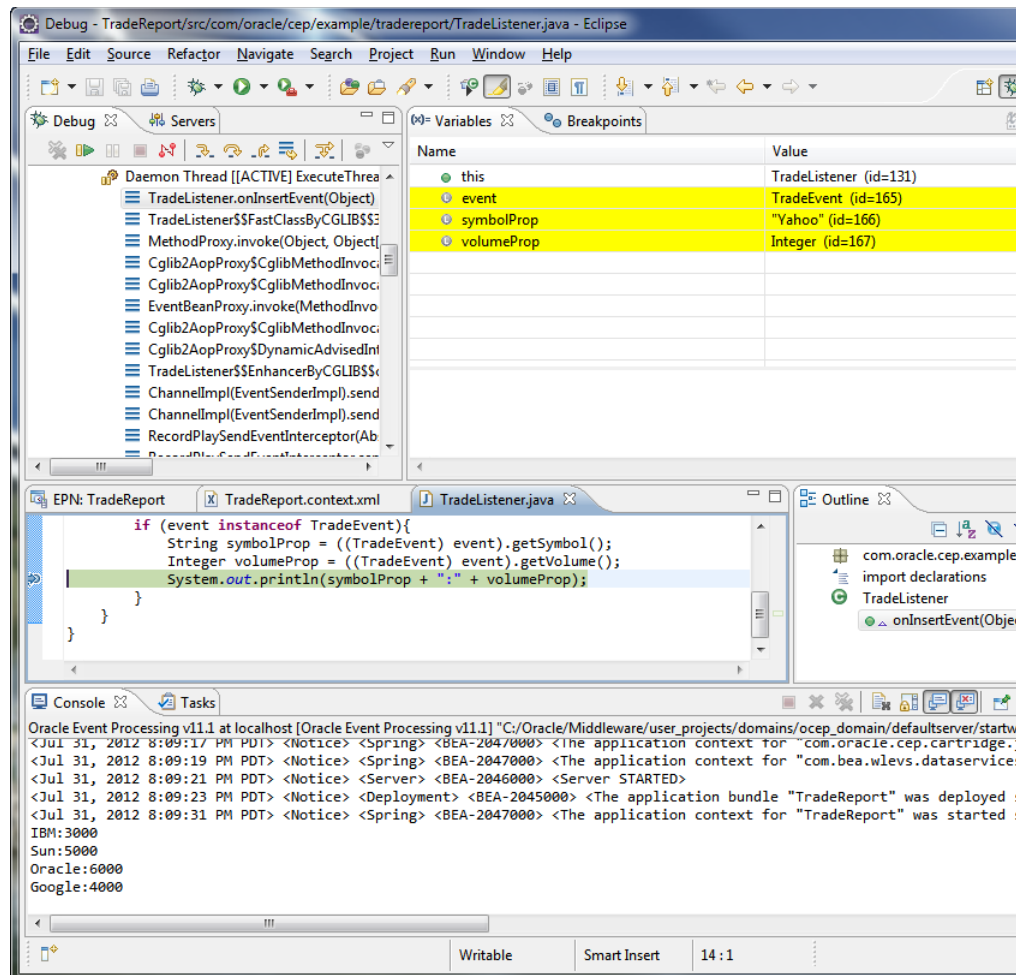
If you see errors when you attempt to republish, you might need to clean and rebuild the project before continuing.

6. Open a command prompt and change directory to the load-generator directory installed with Oracle Event Processing. By default, this is at the following path:

```
ORACLE_HOME/Middleware/ocep_11.1/utills/load-generator
```

7. In the command prompt, type the following, then press **Enter** to start sending event data to your deployed project:
  - On Windows: `runloadgen.cmd StockData.prop`
  - On Linux: `runloadgen.sh StockData.prop`
8. Once the load generator begins sending event data, the IDE should switch to its debugging perspective (if you have the IDE set that way) and pause execution at the breakpoint you set in the Listener class.
9. When execution has paused at the breakpoint, look at the **Variables** view to examine the contents of the event that was received by the listener. By clicking each of the event's properties, you can view the values assigned to each from the CSV file.

10. Click the **Resume** button repeatedly to advance execution from event to event, noticing the values in each.
11. In the **Console** window, notice that the listener code is printing stock symbols and volumes from the events it receives.



12. In the **Servers** view, select the Oracle Event Processing server, then click the **Stop the server** button.
13. In the load generator command prompt window, press CTRL+C to stop the load generator.

That's it! You've created and tested a simple Oracle Event Processing application. In the last step, [Section , "Add an Oracle CQL Processor to Filter Events"](#), you'll make the application a little more interesting by adding some Oracle CQL code.

## Add an Oracle CQL Processor to Filter Events

In this step, you'll add a processor to filter events based on certain criteria. A processor is a stage to which you add Oracle Continuous Query Language (Oracle CQL) code for querying incoming events. Processors and Oracle CQL queries represent much of the real power of event-oriented applications you build with Oracle Event Processing. With Oracle CQL, you can focus the application's logic on just those events you care about, executing sometimes complex logic as events arrive.



If you have used Structured Query Language (SQL), Oracle CQL will appear very familiar. In fact, Oracle CQL is essentially like SQL -- with the same keywords and syntax rules -- but with features added to support the unique aspects of streaming data. (If you aren't familiar with SQL, getting acquainted with it will go a long way toward helping you get the most out of Oracle CQL.)

Remember that event data (and the Oracle Event Processing events that result from it) is a stream of data that the EPN receives sequentially. To continue the comparison of SQL and Oracle CQL, an event may be said in one sense to correspond to a row in a database. However, an important difference is that with events, one event is always before or after another, time-wise, and the stream is potentially infinite and ever-changing. In a relational database, rows may be said to be a finite set where data is relatively static. With a relational database, data is waiting for your query to go and get it; with a stream of event data, data is always flowing into the EPN, where your query examines it as it arrives.

To make the most of the sequential, time-oriented quality of streaming data, CQL includes the ability to:

- Specify a window of a particular time period, or range, from which events should be queried. This could be each five seconds worth of events, for example.
- Specify a window of a particular number of events, called "rows," against which to query. This might be each sequence of 10 events.
- Specify how often the query should execute against the stream by using the `slide` keyword. The query could "slide" every five seconds to a later five-second window of events.
- Separate, or partition, an incoming stream into multiple streams based on particular characteristics of the events. You could have the query create new streams for each of specified stock symbols found in incoming trade events.

In addition, CQL supports common aspects of SQL you might be familiar with, including views and joins. For example, you can write CQL code that performs a join involving streaming event data and data in a relational database table or cache. CQL is extensible through cartridges, with included cartridges providing support for queries that incorporate functionality within Java classes, for calculations specific to spatial data, and to query JDBC data sources.

To get acquainted with processors and CQL, you'll keep the code you're adding here simple. You'll add a query that retrieves certain events fed from the `AdapterOutputChannel`. The query will be designed to retrieve only those trades whose volume is greater than 4000. Events in the query's results will be passed along to the listener.

The following is what the CQL code will look like:

**Example 8-1 GetHighVolume Query Element with CQL Code**

```
<query id="GetHighVolume"><![CDATA[
  select trade.symbol, trade.volume
  from AdapterOutputChannel [now] as trade
  where trade.volume > 4000
]]>
</query>
```

This query can be paraphrased as "for every event coming from the `AdapterOutputChannel` whose trade volume is more than 4000 shares, get the symbol and volume values." The `select`, `from`, and `where` statements should look familiar if you've used SQL. The `now` operator represents a window of time, where the window is

essentially instantaneous -- it includes every event. Another window might have been [range 5], meaning "select from all the events that arrive within each five second period." (That might be useful if you care that certain trades have occurred within five seconds of one another.)

The output of this query will be a relation -- or set, rather than sequence -- that includes all of the events whose trade volume is greater than 4000. Because the window is [now], the set will always have no more than one member. A relation from a larger range might have multiple members in its set.

### Add a **GetHighVolume Processor and Query**

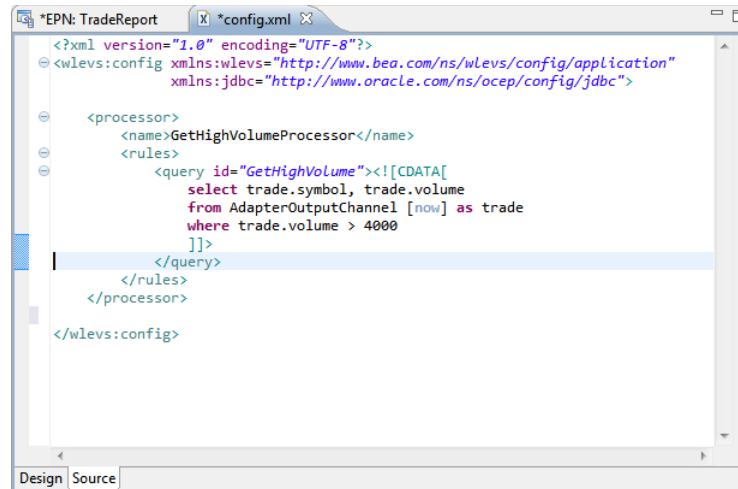
1. In the IDE, in the EPN editor, right-click an empty area of the diagram, then click **New > Processor**.
2. In the **New Processor** dialog, in the **Processor ID** box, enter `GetHighVolumeProcessor`.
3. Select the **Use an existing configuration file** option.  
Remember that you can create a different configuration file for each processor. That might be useful if you have a different person working on each one.
4. In the dropdown, leave `config.xml` selected, then click **OK**.
5. In the EPN editor, notice that a **GetHighVolumeProcessor** icon has been created.
6. Right-click the connector from the **AdapterOutputChannel** icon to the **ListenerBean** icon, then click **Delete**.
7. Click then **AdapterOutputChannel** icon, then drag from it to the **GetHighVolumeProcessor** icon.

Creating this connection makes the processor aware of the channel. After connecting the channel to the processor, you can refer to the channel by its ID value in CQL code.

8. Double-click the **GetHighVolumeProcessor** icon to open its configuration code in the `config.xml` file.
9. In the `config.xml` file, replace the query element with the following query XML:

```
<query id="GetHighVolume"><![CDATA[
    select trade.symbol, trade.volume
    from AdapterOutputChannel [now] as trade
    where trade.volume > 4000
]]>
</query>
```

The code window should look something like the following:



10. In the EPN editor, right-click an empty area of the diagram, then click **New > Channel**.
11. Right-click the icon for the new channel, then click **Rename**.
12. Type `ProcessorOutputChannel` and press **Enter** to rename the channel.
13. Click the **GetHighVolumeProcessor** icon, then drag to the new channel icon to connect the processor and channel.
14. Click the **ProcessorOutputChannel** icon, then drag to the **ListenerBean** icon to connect the channel to the listener.
15. Click the **Layout EPN** button to tidy the diagram so that the icons are in a row from left to right in order of sequence.
16. Double-click the **ProcessorOutputChannel** icon to open the channel's configuration in the assembly XML file.
17. In `TradeReport.context.xml`, replace the default channel configuration with the following XML. In particular, note that you're specifying that `TradeEvent` is the event type that passes through this channel.

```

<wlevs:channel id="ProcessorOutputChannel" event-type="TradeEvent">
  <wlevs:listener ref="ListenerBean" />
</wlevs:channel>

```

18. Save all of the files in the project.

At this point, you should be ready to debug the application again.

### Debug the Project

In this section, you'll debug once more to confirm that your CQL code is producing the results you intend.

1. In the IDE, open the `TradeListener.java` file you created in an earlier step. In the **Project Explorer**, the file should be visible by expanding **TradeReport > src > com.oracle.cep.example.tradereport**.
2. In `TradeListener.java`, confirm that you have a breakpoint at the line with the following code: `System.out.println(symbolProp + ":" + volumeProp);`

You might need to rebuild the project before you can successfully debug.

3. In the **Servers** view, select the server you added: **Oracle Event Processing v11.1 at localhost**
4. With the server selected, click the **Start the server in debug mode** button in the upper right corner of the **Servers** view.

You will likely need to wait for a few moments while the server starts, the IDE compiles your project, and then deploys the TradeReport project as an application to the running server. During this time, the Console view will display status messages related to the server's startup progress. When the Console view's output ends with <Server STARTED>, the server is running in debug mode.

5. In the **Servers** view, confirm that the project does not need to be republished to the server. If the TradeReport entry shows (**Republish**), then right-click the TradeReport entry and click **Force Publish**. After it has been successfully deployed, the entry will show (**Synchronized**).
6. If you don't have one already, open a command prompt and change directory to the load-generator directory installed with Oracle Event Processing. By default, this is at the following path:

ORACLE\_HOME/Middleware/ocep\_v11.1/utills/load-generator

7. In the command prompt, type the following, then press Enter to start sending event data to your deployed project:

- On Windows: `runloadgen.cmd StockData.prop`
- On Linux: `runloadgen.sh StockData.prop`

Once the load generator begins sending event data, the IDE should switch to its debugging perspective (if you have the IDE set that way) and pause execution at the breakpoint you set in the listener class.

8. When execution has paused at the breakpoint, look at the **Variables** view to examine the contents of the event that was received by the listener.

By clicking each of the event's properties, you can view the values assigned to each from the CSV file. Notice that the event contains only two values -- symbol and volume. That's because your query selects only those two values from events that pass through the processor.

9. Click the **Resume** button repeatedly to advance execution from event to event, noticing the values in each.
10. In the **Console** window, notice that the listener code is printing stock symbols and volumes from the events it receives.

If you have coded your query correctly, all of the volume values should be higher than 4000.

11. In the **Servers** view, select the Oracle Event Processing server, then click the **Stop the server** button.
12. In the load generator command prompt window, press CTRL+C to stop the load generator.

That's it -- you've finish this application. For a list of the things covered in this walkthrough, along with links to more information, see [Section , "Summary: Simple Application Walkthrough"](#).

## Summary: Simple Application Walkthrough

This section summarizes the simple application walkthrough, which begins with [Section , "Introduction to the Simple Application Walkthrough"](#). In this walkthrough, you built a simple Oracle Event Processing application. You assembled an event processing network to receive events and report on those that met certain criteria.

For other introductory content, see [Chapter 1, "Overview of Creating Oracle Event Processing Applications"](#).

This walkthrough introduced the most basic aspects of developing Oracle Event Processing applications. Those concepts include:

- IDE features designed to make building Oracle Event Processing applications easier.

To start learning more about the IDE, see [Section , "Overview of Oracle Event Processing IDE for Eclipse"](#).

- Building an application as an event processing network (EPN).

For another introduction to EPNs, see [Section , "How an Oracle Event Processing Application Works"](#).

- Designing event types that model events.

For more information, see [Section , "Overview of Oracle Event Processing Event Types"](#).

- Using adapters to manage interactions with external components.

Oracle Event Processing includes adapters. You can also build your own. See the following sections for more:

- [Chapter 11, "Integrating the Java Message Service"](#)
- [Chapter 12, "Integrating an HTTP Publish-Subscribe Server"](#)
- [Chapter 15, "Integrating an External Component Using a Custom Adapter"](#)
- [Chapter 21, "Testing Applications With the Load Generator and csvgen Adapter"](#)

- Implementing a Java class that can receive or send events.

For more on writing event sinks and event sources, see [Chapter , "Handling Events with Sources and Sinks"](#)

- Event streams and relations as models for handling events as they pass through an event processing network.

For an introduction to streams and relations, see [Section , "Overview of Events, Streams and Relations"](#).

- Using Oracle Continuous Query Language (Oracle CQL) to find and filter events.

For more on Oracle CQL, see [Chapter 17, "Querying an Event Stream with Oracle CQL"](#)



---

---

## Defining and Using Event Types

This chapter describes how to define the Oracle Event Processing event types you will need to carry event data through an event processing network, including how to implement and configure event types and how to access the event type repository.

Through the event types you define, your code has access to event data. As described in this chapter, you define event types that are based on one of several data types such as JavaBean classes you write. You add your event types to your application by configuring them as part of the event type repository. Although event types are typically added to the repository through configuration XML, you can also write code to access the repository programmatically.

This chapter includes the following sections:

- [Overview of Oracle Event Processing Event Types](#)
- [Designing Event Types](#)
- [Creating Event Types](#)
- [Accessing the Event Type Repository](#)
- [Sharing Event Types Between Application Bundles](#)

### Overview of Oracle Event Processing Event Types

An event type is how you represent event data in an Oracle Event Processing application. An event is structured data related to something that occurred at a particular time. For example, if your application is designed to react to changes in a server room's environment, event data could include snapshot information collected by a device that monitors the environment. Or if your application pays attention to trends and patterns related to stock market trades, event data could be values corresponding to a trade, including what stock was traded, what the trade volume was, what the share price was, and so on.

Event data entering your application can arrive in any of a wide variety of forms. By creating an event type to represent the data inside the application, you create a predictable way for your application's logic to work with the data. Because event types are the transport vehicles for event data, defining them is an essential part of building an Oracle Event Processing application.

For a hands-on look at creating and using event types, take a look at [Section , "Create an Event Type to Carry Event Data"](#).

## Where Event Type Instances are Used

Instances of event types you create carry event data through the event processing network (EPN) of your application. Keep in mind how you're going to be using the events in code to make better decisions about designing and implementing event types.

Event type instances are used by either the Oracle Event Processing server or your own application logic. Typically, the Oracle Event Processing server creates an instance of the type and binds the data to it. However, for more control over how the event type instance is created, you can create an event type builder. For more information, see [Section , "Controlling Event Type Instantiation with an Event Type Builder Class"](#).

The following lists the primary places where an event type instance is used:

- When incoming event data arrives from an external source, it is bound to an event type instance.
- When an Oracle CQL processor executes a query and outputs a result, an instance of the event type is created for carrying result event data to a stage that is downstream in the EPN.
- In your application's Java logic, such as Java code for handling events, you can create new event type instances and send them to a stage that is downstream in the EPN. Java code that creates events is known as an event source. For more information about implementing event sources, see [Section , "Implementing an Event Source"](#).

## High-Level Process for Creating Event Types

At a high level, the process for creating an event type is a matter of identifying the event data that the event type will need to carry, then implement the type as one of the data types supported by Oracle Event Processing.

Here are high-level steps:

1. Design the event type. Essentially, this is a matter of identifying the set of event data that's relevant for your application, then choosing how that data should be represented by an event type. For more information, see [Section , "Designing Event Types"](#).
2. Create the event type. Once you know how data should be represented by the type, you can create the type in one of three ways: by implementing it as a `java.lang.Class`, by configuring it as a tuple, or by configuring it as a `java.util.Map`. You might also want to implement an event type builder for more control over how the type is instantiated. For more information, see [Section , "Creating Event Types"](#).

## Designing Event Types

When you design an event type, you map raw event data to the implementation options that Oracle Event Processing supports.

Designing an event includes the following tasks:

- Identify the structure of event data that the event type will represent. This could be the structure of raw event data coming from an external source such as a monitor in a server room. It could also be a data structure required by a downstream stage or component to which your code will send instances of the type. For more information, see [Section , "Identifying the Structure of Event Data"](#).



- Choose the data type that will be the basis of the event type you're creating. Your event type will be based on one of three data types supported by Oracle Event Processing: a JavaBean class, a tuple, or a `java.util.Map` instance. For more information, see [Section , "Choosing a Data Type for an Event Type"](#).
- Keeping in mind the planned uses for the event type, and being aware of potential constraints imposed by those uses. For more information, see [Section , "Constraints on Design of Event Types"](#).

Finally, note that when configuring event types in the application's EPN XML file, you can mix type usage. For more information, see [Section , "Mixing Use of Event Type Data Types"](#).

## Identifying the Structure of Event Data

An early task (though typically a simple one) in defining an event processing network is clarifying the structure of event data, then defining the form in which that data will be handled inside the EPN. The event type you create will include a property for each piece of event data your application cares about. Each of those properties will have its own data type.

Before you can create an event type, use a sample of the event data to define the type's properties and their data types. For an example, consider a very simple set of event data coming from a stock trade. Represented as a row of comma-separated values, the trade event data might look as follows:

```
ORCL,14.1,6000
```

The following table illustrates how you could separate the event data into its distinct values, defining an event type property for each.

Sample Data Value	Role	Type of Data	Event Type Property Name
ORCL	stock symbol	character	symbol
14.1	share price	number	price
6000	volume of shares traded	number	volume

Regardless of how you create your event type, you'll need to have a sense of its properties and their data types. However, how you specify those property data types will vary depending on the data type on which you base your event type. For example, if you implement a JavaBean class as your event type, data types for the three values specified in the table would likely be as follows:

Event Type Property Name	Java Data Type
symbol	String
price	Double
volume	Integer

The data types would be different if you were creating your event type as a tuple or `java.util.Map`. For more information, see [Section , "Choosing a Data Type for an Event Type"](#).

## Choosing a Data Type for an Event Type

Event types you create are based on one of three data types supported by Oracle Event Processing: a JavaBean class, a tuple, or a `java.util.Map`.

Each data type has its own benefits and limitations, but the best practice is to create your event type as a JavaBean class, implementing event type properties as accessor methods. With a JavaBean, you will have greater flexibility to deal with event types as part of your application logic, as well as simplified integration with existing systems. With an event type implemented as a JavaBean class, you can also (if you like) closely control event type instantiation by implementing an event type builder class.

When you create your event type as a tuple or `java.util.Map`, you do so by defining the event type in the EPN assembly file, specifying its properties declaratively. Unless you explicitly declare that a `java.util.Map` should be used, Oracle Event Processing will use the tuple type as the default.

For the benefits and limitations of each supported data type, see [Table 9–1, "Data Types for Event Types"](#):

**Table 9–1 Data Types for Event Types**

Data Type	Description	Benefits and Limitations
JavaBean	A Java class written to JavaBean conventions. In addition to being used by logic you write, the type's accessor ("get" and "set") methods will be used by the Oracle Event Processing server and CQL processor to retrieve and set event property values. For more information, see <a href="#">Section , "Creating an Oracle Event Processing Event Type as a JavaBean"</a> .	<p><b>Benefits:</b> This type is the best practice because it provides the greatest flexibility and ease of use for application logic that handles events. You access property values directly through accessor methods. A JavaBean class is more likely to be useful when integrating your Oracle Event Processing application with other systems. For control over how the type is instantiated, you can implement an event type builder class.</p> <p><b>Limitations:</b> Requires writing a JavaBean class, rather than simply declaring the event type in a configuration file. Oracle CQL does not support JavaBean properties in GROUP BY, PARTITION BY, and ORDER BY, although you can work around this by using an Oracle CQL view.</p>
Tuple	A structure that you create and register declaratively in the EPN assembly file. For more information, see <a href="#">Section , "Creating an Oracle Event Processing Event Type as a Tuple"</a> .	<p><b>Benefits:</b> Requires no Java programming to create the event type. An event type is created by declaring it in the EPN assembly file. Useful for quick prototyping.</p> <p><b>Limitations:</b> Using instances of this type in Java application logic requires programmatically accessing the event type repository to get the instance's property values. A tuple is also unlikely to be useful when integrating the Oracle Event Processing with other systems.</p>
<code>java.util.Map</code>	Based on an instance of <code>java.util.Map</code> . You don't implement or extend the <code>Map</code> interface. Rather, you specify that the interface should be used when configuring the event type in the EPN assembly file. If you write Java code to access the type instance, you treat it as a <code>Map</code> instance. For more information, see <a href="#">Section , "Creating an Oracle Event Processing Event Type as a <code>java.util.Map</code>"</a> .	<p><b>Benefits:</b> Requires no Java programming to create the type. An event type is created by declaring it in the EPN assembly file. Useful for quick prototyping.</p> <p><b>Limitations:</b> Does not perform as well as other types.</p>

For more detailed information on how Oracle CQL handles and supports various data types, see:

- "Datatypes" in the *Oracle Fusion Middleware CQL Language Reference for Oracle Event Processing*

## Constraints on Design of Event Types

Depending on how you plan to use an event type, you might need to keep in mind certain constraints. For example, you might need to limit the data types of its properties or how you set values of certain attributes when configuring the event type.

The following sections describe areas of constraints on event type design:

- [Section , "Constraints on Event Types for Use With the csvgen Adapter"](#)
- [Section , "Constraints on Event Types for Use With a Database Table Source"](#)

### Constraints on Event Types for Use With the csvgen Adapter

When you declaratively specify the properties of an event type for use with the csvgen adapter, you may only use the data types that [Table 9–2](#) describes.

**Table 9–2** *csvgen Adapter Types*

Type	Usage
char	Single or multiple character values. Use for both char and java.lang.String values. Optionally, you may use the length attribute to specify the maximum length of the char value as <a href="#">Example 9–6</a> shows for the property with name id. The default length is 256 characters. If you need more than 256 characters you should specify an adequate length.
int	Numeric values in the range that java.lang.Integer specifies.
float	Numeric values in the range that java.lang.Float 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:

- [Chapter 21, "Testing Applications With the Load Generator and csvgen Adapter"](#)

### Constraints on Event Types for Use With a Database Table Source

You can use a relational database table as a source of event data, binding data from the table to your event type instance at runtime. When your event data source is a database table, you must follow the guidelines represented by the following tables.

When an event type will receive data from a database table, a property configured for the type will each receive data from a particular column in the database. When configuring the event type, note that its property child elements have attributes that have particular meanings and value constraints, as described in [Table 9–3, " EPN Assembly File event-type Element Property Attributes"](#).

**Table 9–3** *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 Event Processing Java type from <a href="#">Table 9–4</a> that corresponds to the column's SQL data type.
length	The column size as specified in the SQL create table statement.

When you specify the properties of an event type for use with a relational database table, you must observe the additional JDBC type restrictions listed in [Table 9–4, " SQL Column Types and Oracle Event Processing Type Equivalents"](#).

**Table 9–4 SQL Column Types and Oracle Event Processing Type Equivalents**

SQL Type	Oracle Event Processing 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	byte[]		Array, of depth 1, of byte.
BIT	java.lang.Boolean	boolean	An instance of java.lang.Boolean.
BLOB	byte[]		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	byte[]		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.
BINARY_DOUBLE <sup>1</sup> or DOUBLE <sup>2</sup>	java.lang.Double	double	An instance of java.lang.Double
BINARY_FLOAT <sup>1</sup> or FLOAT <sup>2</sup>	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	char[]	char	Array, of depth 1, of char.
LONGVARBINARY	byte[]		Array, of depth 1, of byte.
LONGVARCHAR	char[]	char	Array, of depth 1, of char.
NCHAR	char[]	char	Array, of depth 1, of char.
NCLOB	byte[]		Array, of depth 1, of byte.
NUMERIC	java.math.BigDecimal		An instance of java.math.BigDecimal.
NVARCHAR	char[]	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 <i>Oracle Fusion Middleware CQL Language Reference for Oracle Event Processing</i> .
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	byte[]		Array, of depth 1, of byte.
VARCHAR	char[]	char	Array, of depth 1, of char.

<sup>1</sup> Oracle SQL.<sup>2</sup> Non-Oracle SQL.

For more information, see:

- [Section , "Configuring an Oracle CQL Processor Table Source"](#)

## Mixing Use of Event Type Data Types

When defining an event type in the EPN assembly file, you can use a JavaBean-based event type as another event type's property data type.

[Example 9-1](#) shows a tuple event type `Student` that defines its address property as JavaBean event type `Address`.

### Example 9-1 Event Type Repository

```
<event-type-repository>
  <event-type name="Student">
    <property name="name" type="char"/>
    <property name="address" type="Address"/>
  </event-type>

  <event-type name="Address">
    <class-name>test.Address</class-name>
  </event-type>
</event-type-repository>
```

## Creating Event Types

Event types define the event properties that provide access to event data within Oracle Event Processing applications. You then use these event types in the adapter and Java code, as well as in the Oracle CQL rules.

When you create an event type, you choose one of the three supported data types as a base for the type you're creating. The best practice is that you can create the type as a JavaBean class, writing a little Java code to handle event data in properties. You can also create the type declaratively as a tuple or `java.util.Map`.

Before you create types, you might want to see [Section , "Designing Event Types"](#) for design considerations.

This section describes:

- [Section , "Creating an Oracle Event Processing Event Type as a JavaBean"](#)
- [Section , "Creating an Oracle Event Processing Event Type as a Tuple"](#)
- [Section , "Creating an Oracle Event Processing Event Type as a java.util.Map"](#)

## Creating an Oracle Event Processing Event Type as a JavaBean

Creating an event type as a JavaBean gives you the greatest level of flexibility when using instances of the type in your application logic. As a result, using a JavaBean class is the best practice for creating an event type.

---

**Note:** Oracle CQL does not support expressions in `GROUP BY`, `PARTITION BY`, and `ORDER BY`. As a result, these do not support Java properties and will fail. When those features are needed, use Oracle CQL views as a workaround.

---

Part of the flexibility in using a JavaBean event type comes from the two different ways to have the type instantiated: by the server or by an event type builder class that you provide. By providing an event type builder class as an instance factory, you can create the instance yourself, controlling how they are created, including how event data values are bound to JavaBean properties. For more information, see [Section ,](#)

### "Controlling Event Type Instantiation with an Event Type Builder Class".

When creating an event type as a JavaBean class, use the following guidelines to ensure the best integration with the Oracle Event Processing system:

- Create an empty-argument public constructor with which the Oracle Event Processing server can instantiate the class.
- For each event property, implement public accessors (get and set methods) following JavaBean conventions. These will be used by the server to access event property values. When you define an event type as a JavaBean, you may use any Java type for its properties.

If you're unfamiliar with JavaBean conventions, see the JavaBeans Tutorial at <http://docs.oracle.com/javase/tutorial/javabeans/> for additional details.

- Implement the `hashCode` and `equals` methods. This optimizes the class for use by the Oracle Event Processing server, which sometimes uses a hash index whose composite key is created from the event type instance hash codes.

If you're using the Eclipse IDE, you can easily implement `hashCode` and `equals` methods through the Source context menu for a class after you have added its accessors.

- Make the class serializable if you intend to cache events in Oracle Coherence.

This topic describes:

- [Section , "How to Create an Oracle Event Processing Event Type as a JavaBean Using the Event Type Repository Editor"](#)
- [Section , "How to Create an Oracle Event Processing Event Type as a JavaBean Manually"](#)

### How to Create an Oracle Event Processing Event Type as a JavaBean Using the Event Type Repository Editor

This procedure describes how to create and register an Oracle Event Processing event type as a JavaBean using the Oracle Event Processing IDE for Eclipse event type repository editor. For more information about the Oracle Event Processing IDE for Eclipse, see [Example 4, "Overview of the Oracle Event Processing IDE for Eclipse"](#).

Alternatively, you can create and register your event type as a JavaBean manually (see [Section , "How to Create an Oracle Event Processing Event Type as a JavaBean Manually"](#)).

#### To create an Oracle Event Processing event type as a Java Bean using the event type repository editor:

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

Be sure to follow the guidelines described in [Section , "Creating an Oracle Event Processing Event Type as a JavaBean"](#).

[Example 9–2](#) shows the `MarketEvent` which is implemented by the `com.bea.wlevs.example.algotrading.event.MarketEvent` class.

#### **Example 9–2 MarketEvent Class**

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

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() {}

public Double getPrice() {
    return this.price;
}
public void setPrice(Double price) {
    this.price = price;
}

public String getSymbol() {
    return this.symbol;
}
public void setSymbol(String symbol) {
    this.symbol = symbol;
}

public Long getTimestamp() {
    return this.timestamp;
}
public void setTimestamp(Long timestamp) {
    this.timestamp = timestamp;
}

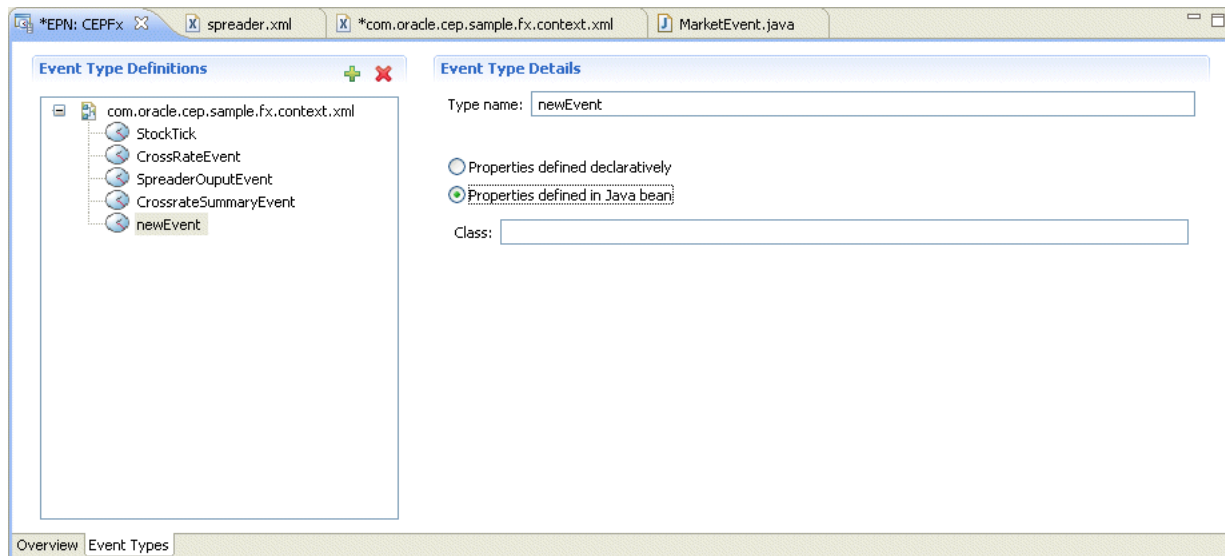
public Long getLatencyTimestamp() {
    return this.latencyTimestamp;
}
public void setLatencyTimestamp(Long latencyTimestamp) {
    this.latencyTimestamp = latencyTimestamp;
}

public Long getVolume() {
    return this.volume;
}
public void setVolume(Long volume) {
    this.volume = volume;
}

// Implementation for hashCode and equals methods.
}
```

2. Compile the JavaBean that represents your event type.
3. In the IDE, open the EPN in the EPN editor.  
For more information, see [Section , "Opening the EPN Editor"](#).
4. Click the **Event Types** tab.
5. Click **Add Event Type** (green plus sign).

A new event is added to the Event Type Definitions list with default name `newEvent` as [Figure 9–1](#) shows.

**Figure 9–1 Event Type Repository Editor - JavaBean Event**

6. In the Event Type Definitions list, select **newEvent**.  
The properties of this event appear in the Event Type Details area as [Figure 9–1](#) shows.
7. Enter a name for this event in the **Type name** field.
8. Click **Properties defined in Java bean**.
9. Enter the fully qualified class name of your JavaBean class in the **Class** field.  
For example `com.bea.wlevs.example.algotrading.event.MarketEvent`.
10. Click the **Save** button in the tool bar (or type CTRL-S).  
The event is now in the event type repository.  
You can use the event type repository editor:
  - a. To view the corresponding event type definition in the EPN assembly file, double-click the event type in the **Event Type Definitions** area.
  - b. To delete this event, select the event type in the Event Type Definitions area and click **Delete Event Type** (red x).

### How to Create an Oracle Event Processing Event Type as a JavaBean Manually

This procedure describes how to create and register an Oracle Event Processing event type as a JavaBean manually.

Alternatively, you can create and register your event type as a JavaBean using the Oracle Event Processing IDE for Eclipse event type repository editor (see [Section , "How to Create an Oracle Event Processing Event Type as a JavaBean Using the Event Type Repository Editor"](#)).

#### To create an Oracle Event Processing event type as a Java bean manually:

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

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



When you design your event, you must restrict your design to the even data types that [Section , "Mixing Use of Event Type Data Types"](#) describes.

[Example 9–3](#) shows the `MarketEvent` which is implemented by the `com.bea.wlevs.example.algotrading.event.MarketEvent` class.

### **Example 9–3 MarketEvent Class**

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

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() {}

    public Double getPrice() {
        return this.price;
    }
    public void setPrice(Double price) {
        this.price = price;
    }

    public String getSymbol() {
        return this.symbol;
    }
    public void setSymbol(String symbol) {
        this.symbol = symbol;
    }

    public Long getTimestamp() {
        return this.timestamp;
    }
    public void setTimestamp(Long timestamp) {
        this.timestamp = timestamp;
    }

    public Long getLatencyTimestamp() {
        return this.latencyTimestamp;
    }
    public void setLatencyTimestamp(Long latencyTimestamp) {
        this.latencyTimestamp = latencyTimestamp;
    }

    public Long getVolume() {
        return this.volume;
    }
    public void setVolume(Long volume) {
        this.volume = volume;
    }

    // Implementation for hashCode and equals methods.
}
```

2. Compile the JavaBean that represents your event type.
3. Register your JavaBean event type in the Oracle Event Processing 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 9–4](#) shows.

**Example 9–4 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 programmatically, use the `EventTypeRepository` class as [Example 9–5](#) shows.

**Example 9–5 Programmatically Registering an Event**

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

For more information, see [Section , "Accessing the Event Type Repository"](#).

**Using JavaBean Event Type Instances in Java Code**

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

The following code implements the `onEvent` method from an event sink class. For more information on event sinks, see [Section , "Implementing an Event Sink"](#).

```
public void onInsertEvent(Object event)
    throws EventRejectedException {
    if (event instanceof MarketEvent){
        MarketEvent marketEvent = (MarketEvent) event;
        System.out.println("Price: " + marketEvent.getPrice());
    }
}
```

**Using JavaBean Event Type Instances in Oracle CQL Code**

The following Oracle CQL rule shows how you can reference the `MarketEvent` in a `SELECT` statement. It assumes an upstream channel called `marketEventChannel` whose event type is `MarketEvent`.

```
<query id="helloworldRule">
  <![CDATA[ SELECT MarketEvent.price FROM marketEventChannel [NOW] ]]>
</query>
```

Also, with property data types implemented as JavaBeans, Oracle CQL code can get values within those properties by using standard JavaBean-style property access.

For example, the following configuration snippet declares a `StudentType` event type that is implemented as a JavaBean class. The `school.Student` class is a JavaBean with an address property that is itself an `Address` JavaBean class. The following query suggests how you might access values of the `Address` object underlying the address property. This query selects student addresses whose postal code begins with "97".

```
<query id="studentAddresses">
  SELECT
    student.address
  FROM
    StudentType as student
  WHERE
    student.address.postalCode LIKE '^97'
```

```
</query>
```

## Controlling Event Type Instantiation with an Event Type Builder Class

You can create an event type builder to have more control over how event type instances are created. For example, using an event type builder you can ensure that the properties of a configured event are correctly bound to the properties of an event type class, such as one you have implemented as a JavaBean. You would need an event type builder in a case, for example, where event property names assumed in CQL code are different from the names of properties declared in the class.

For example, assume the event type has a `firstname` property, but the CQL rule that executes on the event type assumes the property is called `fname`. Assume also that you cannot change either the event type class (because you are using a shared event class from another bundle, for example) or the CQL 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`.

At runtime, an event type builder class receives property values from the Oracle Event Processing server and uses those values to create an instance of the event type class you created. Your event type builder then returns the instance to the server. In this way, your builder class is in effect an intermediary, instantiating event types in cases where the server is unable to determine how to map configured properties to event type properties.

Creating and using an event type builder involves implementing the builder class and configuring a JavaBean event type to use the builder, as described in the following sections:

- [Section , "Implementing an Event Type Builder Class"](#)
- [Section , "Configuring an Event Type that Uses an Event Type Builder"](#)

**Implementing an Event Type Builder Class** 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 Fusion Middleware Java API Reference for Oracle Event Processing* 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 {

    // Called by the server to get an instance of this builder.
    public EventBuilder createBuilder() {
        return new ForeignExchangeBuilder();
    }

    // Inner interface implementation that is the builder.
    static class ForeignExchangeBuilder implements EventBuilder {

        // A Map instance to hold properties until the event type is instantiated.
```

```

private Map<String,Object> values = new HashMap<String,Object>(10);

// Called by the server to put an event type property. Values from the map
// will be used to instantiate the event type.
public void put(String property, Object value) throws IllegalStateException {
    values.put(property, value);
}

// Called by the server to create the event type instance once property
// values have been received.
public Object createEvent() {
    return new ForeignExchangeEvent(
        (String) values.get("symbol"),
        (Double) values.get("price"),
        (String) values.get("fromRate"),
        (String) values.get("toRate"));
}
}
}

```

**Configuring an Event Type that Uses an Event Type Builder** When you register the event type in the EPN assembly file, use the `<wlevs:property name="builderFactory">` child element of the `wlevs:event-type` element to specify the name of the event type builder class. The hard-coded `builderFactory` value of the `name` attribute alerts Oracle Event Processing 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>

```

## Creating an Oracle Event Processing Event Type as a Tuple

You can create an Oracle Event Processing event type as a tuple simply by adding the type's configuration XML to the EPN XML file. As a result, a tuple is the easiest way to create an event type, and so can be useful for quick prototyping. However, both the tuple and `java.util.Map` data types provide less flexibility than creating an event type as a JavaBean class.

With a tuple-based event type, your Java code using instances of the type must always set and get its property values using `EventTypeRepository` APIs.

When you design your event, you must restrict design of the type's properties to the data types described in [Section , "Types for Properties in Tuple-Based Event Types"](#).

Before you get started, consider reading the event type design recommendations in [Section , "Designing Event Types"](#).

This topic describes:

- [Section , "How to Create an Oracle Event Processing Event Type as a Tuple Using the Event Type Repository Editor"](#)
- [Section , "How to Create an Oracle Event Processing Event Type as a Tuple Manually"](#)

## Types for Properties in Tuple-Based Event Types

When you specify the properties of the event type declaratively in the EPN assembly file as a tuple, you may use any of the types specified in `com.bea.wlevs.ede.api.Type`.

For more information on supported property types, see [Section , "wlevs:property"](#).

[Example 9–6](#) shows the use of different types:

### **Example 9–6 Specifying `com.bea.wlevs.ede.api.Type` Data Types for Tuple Event Type Properties**

```
<wlevs:event-type-repository>
  <wlevs:event-type type-name="SimpleEvent">
    <wlevs:properties>
      <wlevs:property name="id" type="char" length="1000" />
      <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 , "Creating an Oracle Event Processing Event Type as a Tuple"](#).

## How to Create an Oracle Event Processing Event Type as a Tuple Using the Event Type Repository Editor

This procedure describes how to create and register an Oracle Event Processing event type as a tuple using the Oracle Event Processing IDE for Eclipse event type repository editor. For more information about the Oracle Event Processing IDE for Eclipse, see [Chapter 4, "Overview of the Oracle Event Processing IDE for Eclipse"](#).

You can instead create and register your event type as a tuple manually (see [Section , "How to Create an Oracle Event Processing Event Type as a Tuple Manually"](#)).

### **To create an Oracle Event Processing event type as a tuple using the event type repository editor:**

1. Decide on the properties your event type requires.

When you design your event, you must restrict your design to the even data types that [Section , "Types for Properties in Tuple-Based Event Types"](#) describes.

2. In the IDE, open the EPN in the EPN editor.

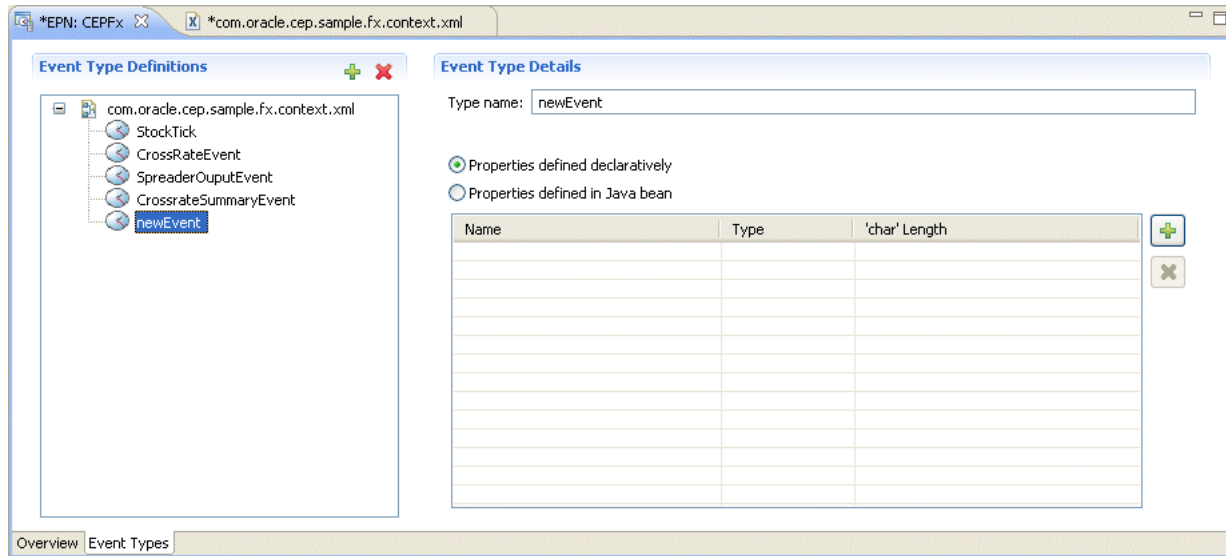
For more information, see [Section , "Opening the EPN Editor"](#).

3. Click the **Event Type** tab.

4. Click **Add Event Type** (green plus sign).

A new event is added to the Event Type Definitions list with default name `newEvent` as [Figure 9–2](#) shows.

**Figure 9–2 Event Type Repository Editor - Tuple Event**



5. In the Event Type Definitions list, select **newEvent**.  
The properties of this event appear in the Event Type Details area as [Figure 9–2](#) shows.
6. Enter a name for this event in the **Type name** field.
7. Click **Properties defined declaratively**.
8. Add one or more event properties:
  - Click **Add Event Property** (green plus sign).  
A new row is added to the Event Type Details table.
  - Click in the **Name** column of this row and enter a property name.
  - Click in the **Type** column of this row and select a data type from the pull down menu.  
When you design your event, you must restrict your design to the even data types that [Section , "Types for Properties in Tuple-Based Event Types"](#) describes.
  - For char data type properties only, click in the **'char' Length** column of this row and enter a value for the maximum length of this char property.  
Optionally, you may used the length attribute to specify the maximum length of the char value. The default length is 256 characters. The maximum length is `java.lang.Integer.MAX_VALUE`. If you need more than 256 characters you should specify an adequate length.
9. Click the **Save** button in the tool bar (or type CTRL-S).  
The event is now in the event type repository.  
You can use the event type repository editor:
  - a. To view the corresponding event type definition in the EPN assembly file, double-click the event type in the **Event Type Definitions** area.
  - b. To delete this event, select the event type in the Event Type Definitions area and click **Delete Event Type** (red x).

## How to Create an Oracle Event Processing Event Type as a Tuple Manually

This procedure describes how to create and register an Oracle Event Processing event type declaratively in the EPN assembly file as a tuple.

Note, however, that the best practice for creating event types is to create them as JavaBean classes. For more information, see [Section , "Creating an Oracle Event Processing Event Type as a JavaBean"](#).

For more information on valid data types, see [Section , "Types for Properties in Tuple-Based Event Types"](#).

### To create an Oracle Event Processing event type as a tuple:

1. Decide on the properties your event type requires.

When you design your event, you must restrict your design to the data types that [Section , "Types for Properties in Tuple-Based Event Types"](#) describes.

2. Register your event type declaratively in the Oracle Event Processing 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 9-7](#) shows.

#### **Example 9-7 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 Event Processing 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 , "Types for Properties in Tuple-Based Event Types"](#).

For reference information on the configuration XML, see [Section , "wlevs:event-type"](#), [Section , "wlevs:properties"](#), and [Section , "wlevs:property"](#)

## Using a Tuple Event Type Instance in Java Code

When using a tuple-based event type in Java code, you must use a `com.bea.wlevs.ede.api.EventTypeRepository` instance to get the names of the type's properties before getting their values. (For information on getting an `EventTypeRepository` instance, see [Section , "Accessing the Event Type Repository"](#).)

The following example uses the repository in a class acting as an event sink.

```
@Service
public void setEventTypeRepository(EventTypeRepository etr) {
    etr_ = etr;
}
...
// Called by the server to pass in the event type instance.
public void onInsertEvent(Object event) throws EventRejectedExpection {

    // Get the event type for the current event instance
```

```

EventType eventType = etr_.getEventType(event);

// Get the event type name
String eventTypeName = eventType.getTypeName();

// Get the event property names
String[] propNames = eventType.getPropertyNames();

// See if property you're looking for is present
if(eventType.isProperty("fromRate")) {
    // Get property value
    Object propValue =
        eventType.getProperty("fromRate").getValue(event);
}
// Throw com.bea.wlevs.ede.api.EventRejectedException to have an
// exception propagated up to senders. Other errors will be
// logged and dropped.
}

```

### Using a Tuple Event Type Instance in Oracle CQL Code

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

```

<query id="FindCrossRatesRule"><![CDATA[
    select ((a.price * b.price) + 0.05) as internalPrice,
           a.fromRate as crossRate1,
           b.toRate as crossRate2
    from FxQuoteStream [range 1] as a, FxQuoteStream [range 1] as b
    where
        NOT (a.price IS NULL)
    and
        NOT (b.price IS NULL)
    and
        a.toRate = b.fromRate
]]></query>

```

## Creating an Oracle Event Processing Event Type as a `java.util.Map`

You can create and register an Oracle Event Processing event type as a `java.util.Map`.

You can create an Oracle Event Processing event type as a `java.util.Map` simply by adding the type's configuration XML to the EPN XML file. As a result, this is an easy way to create an event type. However, both the tuple and `java.util.Map` data types provide less flexibility than creating an event type as a `JavaBean` class.

Creating and using an event type as a `Map` is something of a hybrid of the processes for the tuple type and a `JavaBean` class. As with a tuple-based event type, you create the `Map`-based type declaratively, by configuring it in the EPN XML file. And as with a `JavaBean`-based event type, you needn't use `com.bea.wlevs.ede.api.EventTypeRepository` APIs to access instance property values.

This topic describes:

- [Section , "How to Create an Oracle Event Processing Event Type as a `java.util.Map`"](#)

### Types for Properties in `java.util.Map`-Based Event Types

When you specify the event type properties declaratively in the EPN assembly file as a `java.util.Map`, you may use any Java type for its properties. However, you specify the "type" of the event properties as either:



- The fully qualified name of a Java class that must conform to the same rules as `Class.forName()` and must be available in the application's class loader.
- A Java primitive (for example, `int` or `float`).

You may specify an array by appending the characters `[]` to the event type name.

[Example 9–8](#) shows how to use these types:

#### **Example 9–8 Specifying Java Data Types for `java.util.Map` Event Type Properties**

```
<wlevs:event-type-repository>
  <wlevs:event-type type-name="AnotherEvent">
    <wlevs:properties type="map">
      <wlevs:property>
        <entry key="name" value="java.lang.String"/>
        <entry key="employeeId" value="java.lang.Integer[]"/>
        <entry key="salary" value="float"/>
        <entry key="projectIds" value="short[]"/>
      </wlevs:property>
    </wlevs:properties>
  </wlevs:event-type>
</wlevs:event-type-repository>
```

For more information, see [Section , "How to Create an Oracle Event Processing Event Type as a `java.util.Map`"](#).

#### **How to Create an Oracle Event Processing Event Type as a `java.util.Map`**

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

For more information on valid event type data types, see [Section , "Types for Properties in `java.util.Map`-Based Event Types"](#).

#### **To create an Oracle Event Processing event type as a `java.util.Map`:**

1. Decide on the properties your event type requires.
2. Register your event type in the Oracle Event Processing 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 9–9](#) shows.

#### **Example 9–9 EPN Assembly File event-type-repository**

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

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

- b. To register programmatically, use the `EventTypeRepository` class as [Example 9–10](#) shows.

**Example 9–10 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 [Section , "Accessing the Event Type Repository"](#).

**Using a Map Event Type Instance in Java Code**

When using a Map-based event type instance in Java code, you access its properties as you would with any `java.util.Map` instance.

```
public void onInsertEvent(Object event)
    throws EventRejectedException {

    java.util.Map anEvent = (java.util.Map) event;
    System.out.println("Age: " + anEvent.get("age"));
}
```

**Using a Map Event Type Instance in Oracle CQL Code**

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 age from eventChannel [now] ]]>
</query>
```

## Accessing the Event Type Repository

The Oracle Event Processing event type repository keeps track of the event types defined for your application. When you create an event type, you make it available for use in the application by configuring it in the event type repository. In some cases, you might need to write code that explicitly accesses the repository.

For example, when your event type is created as a tuple, Java logic that accesses instance of the type will need to first retrieve the type definition using the repository API, then use the API to access the instance property values.

The `EventTypeRepository` is a singleton OSGi service. Because it is a singleton, you only need to specify its interface name to identify it. You can get a service from OSGi in any of the following ways:

- [Section , "Using the EPN Assembly File"](#)
- [Section , "Using the Spring-DM @ServiceReference Annotation"](#)
- [Section , "Using the Oracle Event Processing @Service Annotation"](#)

For more information, see *Oracle Fusion Middleware Java API Reference for Oracle Event Processing*.

### Using the EPN Assembly File

You can access the `EventTypeRepository` by specifying an `osgi:reference` in the EPN assembly file as [Example 9–11](#) shows.

**Example 9–11 EPN Assembly File With OSGi Reference to EventTypeRepository**

```
<osgi:reference id="etr" interface="com.bea.wlevs.ede.api.EventTypeRepository" />
<bean id="outputBean" class="com.acme.MyBean" >
  <property name="eventTypeRepository" ref="etr" />
</bean>
```

Then, in the `MyBean` class, you can access the `EventTypeRepository` using the `eventTypeRepository` property initialized by Spring as [Example 9–12](#) shows.

**Example 9–12 Accessing the EventTypeRepository in the MyBean Implementation**

```
package com.acme;

import com.bea.wlevs.ede.api.EventTypeRepository;
import com.bea.wlevs.ede.api.EventType;

public class MyBean {
    private EventTypeRepository eventTypeRepository;

    public void setEventTypeRepository(EventTypeRepository eventTypeRepository) {
        this.eventTypeRepository = eventTypeRepository;
    }

    public void onInsertEvent(Object event) throws EventRejectedException {
        // get the event type for the current event instance
        EventType eventType = eventTypeRepository.getEventType(event);
        // Throw com.bea.wlevs.ede.api.EventRejectedException to have an
        // exception propagated up to senders. Other errors will be
        // logged and dropped.
    }
}
```

**Using the Spring-DM @ServiceReference Annotation**

You can access the `EventTypeRepository` by using the Spring-DM `@ServiceReference` annotation to initialize a property in your Java source as [Example 9–13](#) shows.

**Example 9–13 Java Source File Using the @ServiceReference Annotation**

```
import org.springframework.osgi.extensions.annotation.ServiceReference;
import com.bea.wlevs.ede.api.EventTypeRepository;
...
@ServiceReference
setEventTypeRepository(EventTypeRepository etr) {
    ...
}
```

**Using the Oracle Event Processing @Service Annotation**

You can access the `EventTypeRepository` by using the Oracle Event Processing `@Service` annotation to initialize a property in your Java source as [Example 9–14](#) shows.

**Example 9–14 Java Source File Using the @Service Annotation**

```
import com.bea.wlevs.util.Service;
import com.bea.wlevs.ede.api.EventTypeRepository;
...
@Service
setEventTypeRepository(EventTypeRepository etr) {
    ...
}
```

```
}
```

For more information, see [Section , "com.bea.wlevs.util.Service"](#).

## Sharing Event Types Between Application Bundles

Each Oracle Event Processing 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 Event Processing server) can use these shared event types.

The event type names in this case are scoped to the entire Oracle Event Processing 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.

For more information, see:

- [Section , "Mixing Use of Event Type Data Types"](#)
- [Section , "Assembling Applications With Foreign Stages"](#)
- "Oracle Java Data Cartridge" in the *Oracle Fusion Middleware CQL Language Reference for Oracle Event Processing*

---

## Connecting EPN Stages Using Channels

This chapter describes how to configure Oracle Event Processing channels, event conduits through an event processing network.

This chapter includes the following sections:

- [Overview of Channel Configuration](#)
- [Configuring a Channel](#)
- [Example Channel Configuration Files](#)

### Overview of Channel Configuration

An Oracle Event Processing 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).

You may use a channel with both streams and relations. For more information, see [Section , "Channels Representing Streams and Relations"](#).

When you create a channel in your Event Processing Network (EPN), it has a default configuration. For complete details, see [Section , "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 10–1](#), the corresponding component configuration file channel element is shown in [Example 10–2](#).

#### **Example 10–1 EPN Assembly File Channel Id: priceStream**

```
<wlevs:channel id="priceStream" event-type="PriceEvent">
  <wlevs:listener ref="filterFanoutProcessor" />
  <wlevs:source ref="PriceAdapter" />
</wlevs:channel>
```

#### **Example 10–2 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 Event Processing 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 Event Processing IDE for Eclipse creates one component configuration file and one EPN assembly file.

Component configuration files are deployed as part of the Oracle Event Processing application bundle. You can later update this configuration at runtime using Oracle Event Processing Visualizer, the `wlevs.Admin` utility, or manipulating the appropriate JMX Mbeans directly.

This section describes:

- [Section , "When to Use a Channel"](#)
- [Section , "Channels Representing Streams and Relations"](#)
- [Section , "System-Timestamped Channels"](#)
- [Section , "Application-Timestamped Channels"](#)
- [Section , "Controlling Which Queries Output to a Downstream Channel: selector"](#)
- [Section , "Batch Processing Channels"](#)
- [Section , "EventPartitioner Channels"](#)

For more information, see:

- [Section , "Overview of Component Configuration Files"](#)
- [Section , "Creating EPN Assembly Files"](#)
- *Oracle Fusion Middleware Visualizer User's Guide for Oracle Event Processing*
- "wlevs.Admin Command-Line Reference" in the *Oracle Fusion Middleware Administrator's Guide for Oracle Event Processing*
- "Configuring JMX for Oracle Event Processing" in the *Oracle Fusion Middleware Administrator's Guide for Oracle Event Processing*

## When to Use a Channel

When constructing your EPN, consider the following rules:

- A channel is mandatory when connecting an Oracle CQL processor to a down-stream stage.
- A channel is mandatory when connecting a push source stream or relation to a processor.

A channel is mandatory for a push source because in that case, the Oracle CQL processor does need to be aware of its shape (that is, DDL is required) and so does need the channel to act as intermediary.

- A channel is optional when connecting an external relation, or pull source, such as a cache or table source, to a processor.

A channel is not needed between a pull source, such as a cache or table, and a processor because the pull source represent an external relation. For an external relation, the only valid operation is a join between a stream and a `NOW` window operator and hence it is considered a pull source. In other words, the join actually happens outside of the Oracle CQL processor. Because it is a pull, the Oracle CQL processor does not need to be aware of its shape (that is, no DDL is required) and so does not need the channel to act as intermediary.

In general, use a channel between components when:

- Buffering is needed between the emitting component and the receiver.
- Queuing or concurrency is needed for the receiving component.
- If a custom adapter is used and thread control is necessary.

It is a good design practice to include channels in your EPN to provide the flexibility of performance tuning (using buffering, queuing, and concurrency options) later in the design lifecycle. Setting the channel attribute `max-threads` to 0 puts a channel in pass-through mode and incurs no performance penalty.

For more information, see:

- [Section , "EventPartitioner Channels"](#)
- [Table C-9, " Attributes of the wlevs:channel Application Assembly Element"](#)

## Channels Representing Streams and Relations

A channel can represent either a stream or a relation.

For more information, see:

- [Section , "Implementing RelationSource"](#)
- [Section , "Overview of Channel Configuration"](#)
- [Section , "Overview of Events, Streams and Relations"](#)

### Channels as Streams

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).

### Channels as Relations

A relation supports inserts, deletes, and updates. You specify a channel as a relation by setting EPN assembly element `wlevs:channel` attribute `is-relation` to `true`.

When a channel is a relation, you must specify one or more event properties that define event identity using the `wlevs:channel` attribute `primary-key` as [Example 10-3](#) shows.

#### **Example 10-3 Channel as Relation: primary-key Attribute**

```
...
<wlevs:channel id="priceStream" event-type="PriceEvent" primary-key="stock,broker">
  <wlevs:listener ref="filterFanoutProcessor" />
</wlevs:channel>
```

```

    <wlevs:source ref="PriceAdapter" />
</wlevs:channel>
...

```

#### Example 10–4 PriceEvent

```

<wlevs:event-type-repository>
  <wlevs:event-type type-name="PriceEvent">
    <wlevs:property>
      <entry key="stock" value="java.lang.String" />
      <entry key="broker" value="java.lang.String" />
      <entry key="high" value="float" />
      <entry key="low" value="float" />
    </wlevs:property>
  </wlevs:event-type>
</wlevs:event-type-repository>

```

For more information, see `primary-key` in [Table C–9, "Attributes of the `wlevs:channel` Application Assembly Element"](#).

## System-Timestamped Channels

By default, channels are system-timestamped. In this case, Oracle Event Processing 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 , "How to Configure a System-Timestamped Channel Using Oracle Event Processing IDE for Eclipse"](#)
- [Section , "heartbeat"](#)

## 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 configurable `wlevs:expression` element. 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 the `StreamSender.sendHeartbeat` method to send an event of type `heart-beat` downstream to `StreamSink` listeners in the EPN.

For more information, see:

- [Section , "How to Configure an Application-Timestamped Channel Using Oracle Event Processing IDE for Eclipse"](#)
- [Section , "wlevs:application-timestamped"](#)
- [Section , "wlevs:expression"](#)

## Controlling Which Queries Output to a Downstream Channel: selector

If you configure an Oracle CQL processor with more than one query, by default, all queries output their results to the downstream channel.

You can control which queries may output their results to a downstream channel using the channel `selector` child element. Note that you must use a selector when you have multiple channels connected on the downstream side of a processor. This is because



the output of a query in the processor has a particular type; selecting for a specific query ensures that the query output is accepted by the channel.

[Figure 10-1](#) shows an EPN with channel `filteredStream` connected to up-stream Oracle CQL processor `filterFanoutProcessor`.

**Figure 10-1 EPN With Oracle CQL Processor and Down-Stream Channel**



[Example 10-5](#) shows the queries configured for the Oracle CQL processor.

**Example 10-5 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 10-5](#) shows, then, by default, all query results are output to the processor's out-bound channel (`filteredStream` in [Figure 10-1](#)). Optionally, in the component configuration source, you can use the channel element selector child element to specify a space-delimited list of one or more Oracle CQL query names that may output their results to the channel as [Example 10-6](#) shows. In this example, query results for query `Yr3Sector` and `Yr2Sector` are output to `filteredStream` but not query results for query `Yr1Sector`.

**Example 10-6 Using selector to Control Which Query Results are Output**

```

<channel>
  <name>filteredStream</name>
  <selector>Yr3Sector Yr2Sector</selector>
</channel>
  
```

You may configure a channel element with a selector before creating the queries in the upstream processor. In this case, you must specify query names that match the names in the selector.

---

**Note:** The selector child element is only applicable if the up-stream node is an Oracle CQL processor. For more information, see [Chapter 17, "Querying an Event Stream with Oracle CQL"](#).

---

For more information, see [Appendix , "selector"](#).

## Batch Processing Channels

By default, a channel processes events as they arrive. Alternatively, you can configure a channel to batch events together that have the same timestamp and were output from the same query by setting the `wlevs:channel` attribute `batching` to `true` as [Example 10-7](#) shows.

### **Example 10-7** Batch Processing Channel

```
...
<wlevs:channel id="priceStream" event-type="PriceEvent" batching="true">
  <wlevs:listener ref="filterFanoutProcessor" />
  <wlevs:source ref="PriceAdapter" />
</wlevs:channel>
...
```

For more information, see:

- [Section , "Implementing RelationSource"](#)
- `batching` in [Table C-9, "Attributes of the wlevs:channel Application Assembly Element"](#)

## EventPartitioner Channels

By default, a channel broadcasts each event to every listener.

When you configure a channel to use an `EventPartitioner`, each time an incoming event arrives, the channel selects a listener and dispatches the event to that listener instead of broadcasting each event to every listener.

You can use an `EventPartitioner` on a channel to improve scalability.

For more information, see [Section , "EventPartitioner"](#).

## Handling Faults in Channels

You can write code to handle exceptions that occur in stages that are downstream of a channel, then thrown to the channel. By default, the fault-handling behavior for a channel is as follows:

- If the channel's `max-threads` setting is 0 (also known as a pass-through channel), then the exception is re-thrown to the next upstream stage in the EPN.
- If the channel's `max-threads` setting is greater than 0, then the exception is logged and dropped, along with any events associated with the fault.

You can write a fault handling class and associate the handler with channels whose `max-threads` values are greater than 0. With a fault handler associated with the channel, exceptions thrown to the channel are received by the handler, where your code can handle the fault or re-throw it. Keep in mind that if your fault handling code re-throws the exception, the exception is logged but events related to the exception are lost. If you want to keep track of events involved in these exceptions, you must persist them with your code, such as by writing the event data to a data source connected to your EPN.

For information on writing fault handlers, see [Section , "Handling Faults"](#).

## Configuring a Channel

You can configure a channel manually or by using the Oracle Event Processing IDE for Eclipse.

See [Section , "Component Configuration Schema wlevs\\_application\\_config.xsd"](#) for the complete XSD Schema that describes the channel component configuration file.

See [Section , "Example Channel Configuration Files"](#) for a complete example of a channel configuration file.

This section describes the following topics:

- [Section , "How to Configure a System-Timestamped Channel Using Oracle Event Processing IDE for Eclipse"](#)
- [Section , "How to Configure an Application-Timestamped Channel Using Oracle Event Processing IDE for Eclipse"](#)
- [Section , "How to Create a Channel Component Configuration File Manually"](#)

### How to Configure a System-Timestamped Channel Using Oracle Event Processing 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 Event Processing IDE for Eclipse. Optionally, you can create a channel configuration file manually (see [Section , "How to Create a Channel Component Configuration File Manually"](#)).

For more information, see:

- [Section , "System-Timestamped Channels"](#)
- [Chapter 7, "Oracle Event Processing IDE for Eclipse and the Event Processing Network"](#)

#### To configure a channel using Oracle Event Processing IDE for Eclipse:

1. Use Oracle Event Processing IDE for Eclipse to create a channel.  
See [Section , "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.

Required attributes include:

- `id`
- `event-type`

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`.

If you specify this channel as a relation, you must also configure the channel attribute `primary-key`.

See [Table C-9, "Attributes of the wlevs:channel Application Assembly Element"](#).

- c. Add the appropriate `wlevs:channel` child elements.
  - [Appendix, "wlevs:instance-property"](#)
  - [Appendix, "wlevs:listener"](#)
  - [Appendix, "wlevs:property"](#)
  - [Appendix, "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 Event Processing 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 10-8](#). Otherwise, proceed to step 7.

**Example 10-8 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 10-9](#) shows.

**Example 10-9 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 10–10](#), the corresponding configuration file channel element is shown in [Example 10–11](#).

**Example 10–10 EPN Assembly File Channel Id: priceStream**

```

<wlevs:channel id="priceStream" event-type="PriceEvent">
  <wlevs:listener ref="filterFanoutProcessor" />
  <wlevs:source ref="PriceAdapter" />
</wlevs:channel>

```

**Example 10–11 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 Event Processing 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.

With multiple threads enabled, com.bea.wlevs.ede.api.EventRejectedException instances thrown from downstream components in the EPN won't be propagated up to an adapter from which the channel is receiving events unless the channel has an associated fault handler. For more information, see [Section , "Handling Faults in Channels"](#).

- 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>
```

Note that when a queued channel becomes full, the default behavior is to discard events that could not be inserted into the queue because the queue is full. You can configure the channel so that it will instead raise an `EventProcessingException` to its upstream stage. The exception contains the events that could not be inserted into the queue. To support this behavior, set the `fail-when-rejected` setting to true. You can also specify a timeout value after which events are dropped or an exception is raised. For configuration reference information, see ["fail-when-rejected"](#) and ["offer-timeout"](#).

- Add a `heartbeat` child element to specify the number of nanoseconds a channel can be idle before Oracle Event Processing generates a heartbeat event to advance time. The default is 100,000,000 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.

```
<channel>
  <name>MatchOutputChannel</name>
  <heartbeat>10000</heartbeat>
</channel>
```

- Add a `selector` child element to specify which up-stream Oracle CQL processor queries are permitted to output their results to the channel.

You may configure a channel element with a `selector` before creating the queries in the upstream processor. In this case, you must specify query names that match the names in the `selector`.

For more information, see [Section , "Controlling Which Queries Output to a Downstream Channel: selector"](#).

---

**Note:** The `selector` attribute is only applicable if the up-stream node is an Oracle CQL processor. For more information, see [Chapter 17, "Querying an Event Stream with Oracle CQL"](#).

---

## 10. Select **File > Save**.

The EPN Editor adds a configuration badge to the channel as [Figure 10–2](#) shows. For more information, see [Section , "Configuration Badging"](#).

**Figure 10–2 Channel With Configuration Badge**



## How to Configure an Application-Timestamped Channel Using Oracle Event Processing 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 Event Processing IDE for Eclipse. Optionally, you can create a channel configuration file manually (see [Section , "How to Create a Channel Component Configuration File Manually"](#)).

For more information, see:

- [Section , "Application-Timestamped Channels"](#)
- [Chapter 7, "Oracle Event Processing IDE for Eclipse and the Event Processing Network"](#)

### To configure a channel using Oracle Event Processing IDE for Eclipse:

1. Use Oracle Event Processing IDE for Eclipse to create a channel.  
See [Section , "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 C–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 Event Processing 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 , "wlevs:application-timestamped"](#).
  - d. Add other appropriate `wlevs:channel` child elements.
    - [Appendix , "wlevs:instance-property"](#)
    - [Appendix , "wlevs:listener"](#)
    - [Appendix , "wlevs:property"](#)
    - [Appendix , "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 Event Processing 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 10–8](#). Otherwise, proceed to step 7.

**Example 10–12 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 10–9](#) shows.

**Example 10–13 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 10–10](#), the corresponding configuration file `channel` element is shown in [Example 10–11](#).



**Example 10–14 EPN Assembly File Channel Id: priceStream**

```
<wlevs:channel id="priceStream" event-type="PriceEvent">
  <wlevs:listener ref="filterFanoutProcessor" />
  <wlevs:source ref="PriceAdapter" />
</wlevs:channel>
```

**Example 10–15 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 Event Processing 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.

Note that with multiple threads enabled, `com.bea.wlevs.ede.api.EventRejectedException` instances thrown from downstream components in the EPN won't be propagated up to an adapter from which the channel is receiving events.

- 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>
```

Note that when a queued channel becomes full, the default behavior is to discard events that could not be inserted into the queue because the queue is full. You can configure the channel so that it will instead raise an `EventProcessingException` to its upstream stage. The exception contains the events that could not be inserted into the queue. To support this behavior, set the `fail-when-rejected` setting to true. You can also specify a timeout value after which events are dropped or an exception is raised. For configuration reference information, see ["fail-when-rejected"](#) and ["offer-timeout"](#).

- Add a `selector` child element to specify which up-stream Oracle CQL processor queries are permitted to output their results to the channel.

You may configure a channel element with a `selector` before creating the queries in the upstream processor. In this case, you must specify query names that match the names in the `selector`.

For more information, [Section , "Controlling Which Queries Output to a Downstream Channel: selector"](#).

---

**Note:** The `selector` attribute is only applicable if the up-stream node is an Oracle CQL processor. For more information, see [Chapter 17, "Querying an Event Stream with Oracle CQL"](#).

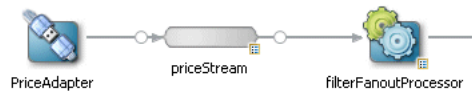
---

For more information, [Section , "selector"](#).

#### 10. Select **File > Save**.

The EPN Editor adds a configuration badge to the channel as [Figure 10–3](#) shows. For more information, see [Section , "Configuration Badging"](#).

**Figure 10–3 Channel With Configuration Badge**



## How to Create a Channel Component Configuration File Manually

Although the Oracle Event Processing IDE for Eclipse is the most efficient and least error-prone way to create and a channel configuration file (see [Section , "How to Configure a System-Timestamped Channel Using Oracle Event Processing 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 , "Creating EPN Assembly Files"](#) 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 C–9, "Attributes of the wlevs:channel Application Assembly Element"](#).

- b. Add the appropriate `wlevs:channel` child elements.

- [Appendix , "wlevs:application-timestamped"](#)
- [Appendix , "wlevs:instance-property"](#)

- [Appendix , "wlevs:listener"](#)
  - [Appendix , "wlevs:property"](#)
  - [Appendix , "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 Event Processing knows to which particular channel component in the EPN assembly file this channel configuration applies. See [Section , "Creating EPN Assembly Files"](#) for details.

For example, if your application has two streams, the configuration file might initially look like:

```
<?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:

```
<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 Event Processing 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.

Note that with multiple threads enabled, `com.bea.wlevs.ede.api.EventRejectedException` instances thrown from downstream components in the EPN won't be propagated up to an adapter from which the channel is receiving events.

- 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>
```

Note that when a queued channel becomes full, the default behavior is to discard events that could not be inserted into the queue because the queue is full. You can configure the channel so that it will instead raise an `EventProcessingException` to its upstream stage. The exception contains the events that could not be inserted into the queue. To support this behavior, set the `fail-when-rejected` setting to true. You can also specify a timeout value after which events are dropped or an exception is raised. For configuration reference information, see ["fail-when-rejected"](#) and ["offer-timeout"](#).

- Add a `heartbeat` child element to specify a new number of nanoseconds a channel can be idle before Oracle Event Processing generates a heartbeat event to advance time. The default is 100,000,000 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.

```
<channel>
  <name>MatchOutputChannel</name>
  <heartbeat>10000</heartbeat>
</channel>
```

- Add a `selector` child element to specify which up-stream Oracle CQL processor queries are permitted to output their results to the channel.

You may configure a channel element with a selector before creating the queries in the upstream processor. In this case, you must specify query names that match the names in the selector.

For more information, [Section , "Controlling Which Queries Output to a Downstream Channel: selector"](#).

---

**Note:** The selector attribute is only applicable if the up-stream node is an Oracle CQL processor. For more information, see [Chapter 17, "Querying an Event Stream with Oracle CQL"](#).

---

For more information, [Section , "selector"](#).

6. Save and close the configuration file.

## Example Channel Configuration Files

[Figure 10–4](#) 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 10–4](#)).

**Figure 10–4 EPN with Two Channels**



This section provides example channel configuration files, including:

- [Section , "Channel Component Configuration File"](#)
- [Section , "Channel EPN Assembly File"](#)

## Channel Component Configuration File

[Example 10–16](#) shows a sample component configuration file that configures the two channels shown in [Figure 10–4](#).

**Example 10–16 Sample Channel Component Configuration File**

```

<?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>
</nl:config>

```

## Channel EPN Assembly File

[Example 10–17](#) shows a EPN assembly file that configures the two channels shown in [Figure 10–4](#).

### **Example 10–17 Channel EPN Assembly File**

```

<?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_1_1_6.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:properties>
        </wlevs:event-type>
    </wlevs:event-type-repository>

```

```

        <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="eventName"
        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="bidAskBBASStream" />
</wlevs:processor>

<wlevs:processor id="analyticsProcessor">
    <wlevs:listener ref="bidAskAnalyticsStream" />
</wlevs:processor>

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



---

---

## Integrating the Java Message Service

This chapter describes how to use JMS adapters to connect the Java Message Service with an Oracle Event Processing event processing network to receive and send JMS messages.

This chapter includes the following sections:

- [Overview of JMS Adapter Configuration](#)
- [Configuring a JMS Adapter for a JMS Service Provider](#)
- [Creating a Custom Converter Between JMS Messages and Event Types](#)
- [Encrypting Passwords in the JMS Adapter Component Configuration File](#)
- [Configuring the JMS Adapter EPN Assembly File](#)
- [Configuring the JMS Adapter Component Configuration File](#)

### Overview of JMS Adapter Configuration

The Oracle Event Processing JMS adapters support any Java EE compliant JMS service provider that provides a Java client. For more information, see [Section , "JMS Service Providers"](#).

Oracle Event Processing provides the following Java Message Service (JMS) adapters that you can use in your event applications to send and receive messages to and from a JMS destination, respectively, without writing any Java code:

- [Section , "Inbound JMS Adapter"](#)
- [Section , "Outbound JMS Adapter"](#)

For general information about JMS, see Java Message Service on the Sun Developer Network at <http://java.sun.com/products/jms/>.

### JMS Service Providers

The Oracle Event Processing JMS adapters support any Java EE compliant JMS service provider that provides a Java client.

This chapter describes how to configure the Oracle Event Processing JMS inbound and outbound adapters for use with the following JMS service providers:

- [Section , "How to Configure a JMS Adapter for Oracle WebLogic Server JMS Manually"](#)
- [Section , "How to Configure a JMS Adapter for Tibco EMS JMS Manually"](#)

If your JMS service provider is not in this list, you can still configure Oracle Event Processing JMS adapters for use with your JMS service provider. Review the procedure in [Section , "How to Configure a JMS Adapter for Tibco EMS JMS Manually"](#) as a configuration model, consult your JMS service provider documentation, and adapt this procedure to suit your JMS service provider.

For more information, see [Section , "Configuring a JMS Adapter for a JMS Service Provider"](#).

## Inbound JMS Adapter

The inbound JMS adapter receives messages from a JMS destination and converts them into events.

You specify an inbound JMS adapter in the EPN assembly file as follows:

```
...
  <wlevs:adapter id="myJmsInbound" provider="jms-inbound">
    ...
  </wlevs:adapter>
...
```

You configure an inbound JMS adapter in its component configuration file as follows:

```
...
  <jms-adapter>
    <name>myJmsInbound</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>
...
```

This section describes:

- [Section , "Conversion Between JMS Messages and Event Types"](#)
- [Section , "Single and Multi-threaded Inbound JMS Adapters"](#)
- [Section , "Configuring a JMS Adapter for Durable Subscriptions"](#)

For more information, see:

- [Section , "JMS Inbound Adapter EPN Assembly File Configuration"](#)
- [Section , "JMS Inbound Adapter Component Configuration"](#)

### Conversion Between JMS Messages and Event Types

By default, the inbound JMS adapter automatically converts JMS messages into events by matching property names with a specified event type if the following is true:

- You must specify an event type using the `jms-adapter` element `event-type` child element in the JMS adapter component configuration file.

The JMS adapter converts incoming JMS messages into the Oracle Event Processing event type this element specifies.

- JMS messages must 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.

Optionally, you may customize JMS message to event type 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. In this case, you do not specify an event type using the `jms-adapter` element `event-type` child element in the JMS adapter component configuration file.

For more information, see [Section , "Creating a Custom Converter Between JMS Messages and Event Types"](#).

### Single and Multi-threaded Inbound JMS Adapters

By default, an inbound JMS adapter is single-threaded. That is, the inbound JMS adapter uses a single thread to read messages from the JMS destination.

When the inbound JMS adapter is single-threaded, event order is guaranteed.

To improve scalability, you can configure an inbound JMS adapter to use multiple threads to read messages from the JMS destination. The simplest way to do this is to configure the adapter with a work manager. You can specify a dedicated work manager used only by the adapter or you can share a work manager amongst several components such as other adapters and Jetty.

When the inbound JMS adapter is multi-threaded, event order is not guaranteed.

For more information, see:

- [work-manager](#) and [concurrent-consumers](#) in [Table 11–1, "jms-adapter Inbound Child Elements"](#)
- [Section , "EventPartitioner"](#)

### Configuring a JMS Adapter for Durable Subscriptions

You can configure an inbound JMS adapter to be a client in a durable subscription to a JMS topic. A durable subscription ensures that the adapter receives published messages even if the adapter becomes inactive. When the inbound adapter connects to the JMS server it will register the durable subscription, and subsequent messages sent to the topic will be retained during periods when the subscriber is disconnected (unless they expire) and delivered when it reconnects.

A durable subscription assumes that the publisher that is publishing JMS messages to the topic is using the persistent delivery mode. Note that publisher might be the Oracle Event Processing outbound JMS adapter (in other words, its `delivery-mode` value must be `persistent`, the default value).

To create a durable subscription in the adapter you do the following:

- Ensure that the JMS message publisher is delivering messages in persistent mode.
- Specify a client ID for the connection factory. On Oracle WebLogic Server, the client ID can be set on the connection factory administratively using the console. Note that this implies that you should have a dedicated connection factory configured for each adapter instance that is using durable subscribers.
- Set three `jms-adapter` properties:
  - `destination-type` must be set to `TOPIC`.
  - `durable-subscription` must be set to `true`.

- `durable-subscription-name`  is set to a unique identifier for the subscription.

For more information about these properties, see [Section , "JMS Inbound Adapter Component Configuration"](#).

## Outbound JMS Adapter

The outbound JMS adapter sends events to a JMS destination, 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 Event Processing take care of the conversion between messages and event types, the following is true:

- You must specify an event type using the  `jms-adapter`  element  `event-type`  child element in the JMS adapter component configuration file.

The JMS adapter converts incoming JMS messages into the Oracle Event Processing event type this element specifies.

- 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` .

You specify an outbound JMS adapter in the EPN assembly file as follows:

```
...
<wlevs:adapter id="myJmsOutbound" provider="jms-outbound">
  ...
</wlevs:adapter>
...
```

You configure an outbound JMS adapter in its component configuration file as follows:

```
...
<jms-adapter>
  <name>myJmsOutbound</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>
...
```

For more information, see:

- [Section , "JMS Outbound Adapter EPN Assembly File Configuration"](#)
- [Section , "JMS Outbound Adapter Component Configuration"](#)
- [Section , "Creating a Custom Converter Between JMS Messages and Event Types."](#)

## Configuring a JMS Adapter for a JMS Service Provider

This section describes how to configure the Oracle Event Processing JMS inbound and outbound adapters:

- [Section , "How to Configure a JMS Adapter Using the Oracle Event Processing IDE for Eclipse"](#)
- [Section , "How to Configure a JMS Adapter Manually"](#)

This section provides examples specific to the following JMS service providers:

- [Section , "How to Configure a JMS Adapter for Oracle WebLogic Server JMS Manually"](#)
- [Section , "How to Configure a JMS Adapter for Tibco EMS JMS Manually"](#)

If your JMS service provider is not in this list, and your JMS service provider offers a Java client, then you can still configure Oracle Event Processing JMS adapters for use with your JMS service provider. Review the procedure in [Section , "How to Configure a JMS Adapter for Tibco EMS JMS Manually"](#) as a configuration model, consult your JMS service provider documentation, and adapt this procedure to suit your JMS service provider.

For more information, see [Section , "JMS Service Providers"](#).

## How to Configure a JMS Adapter Using the Oracle Event Processing IDE for Eclipse

The simplest way to create and configure a JMS adapter is using the Oracle Event Processing IDE for Eclipse adapter wizard.

For more information, see [Section , "How to Create an Adapter Node"](#).

After using the adapter wizard to create and specify the basic JMS adapter configuration, review [Section , "How to Configure a JMS Adapter Manually"](#) to complete the configuration.

## How to Configure a JMS Adapter Manually

This section describes how to create and configure a JMS adapter manually. It describes the detailed steps that you may require depending on your JMS adapter application and service provider.

The simplest way to create and configure a JMS adapter is using the Oracle Event Processing IDE for Eclipse adapter wizard as [Section , "How to Configure a JMS Adapter Using the Oracle Event Processing IDE for Eclipse"](#) describes. After using the adapter wizard to create and specify the basic JMS adapter configuration, review this procedure to complete the configuration.

### To configure a JMS adapter manually:

1. In the EPN assembly file of the application, add a `wlevs:adapter` element for each inbound and outbound JMS adapter you want to use in your application.

[Example 11-1](#) shows the `wlevs:adapter` element for a JMS inbound adapter.

#### **Example 11-1** *wlevs:adapter Element for Inbound Adapter*

```
<wlevs:adapter id="inboundJmsAdapter1" provider="jms-inbound">
...
</wlevs:adapter>
```

See:

- [Section , "JMS Inbound Adapter EPN Assembly File Configuration"](#)
  - [Section , "JMS Outbound Adapter EPN Assembly File Configuration"](#)
2. In the component configuration file of the application, add a `jms-adapter` element for each inbound and outbound JMS adapter you want to use in your application.

[Example 11-2](#) shows the `jms-adapter` element for the JMS inbound adapter in [Example 11-1](#).

**Example 11–2 *jms-adapter* Element for Inbound Adapter**

```
<jms-adapter>
  <name>inboundJmsAdapter1</name>
  ...
</jms-adapter>
```

For each `jms-adapter` element, the `name` child element must be set to the corresponding `wlevs:adapter` element `id` child element.

See:

- [Section , "JMS Inbound Adapter Component Configuration"](#)
  - [Section , "JMS Outbound Adapter Component Configuration"](#)
3. Decide how you want to convert between JMS messages and Oracle Event Processing event types:

- a. If you want the JMS adapters to perform automatic conversion, specify an event type using the `jms-adapter` element `event-type` child element in the JMS adapter component configuration file.

See:

- [Section , "JMS Inbound Adapter Component Configuration"](#)
- [Section , "JMS Outbound Adapter Component Configuration"](#)

- b. If you want the JMS adapters to perform custom conversion, create a custom converter Java class and register it in the EPN assembly file.

See [Section , "Creating a Custom Converter Between JMS Messages and Event Types"](#).

4. Configure the `jms-adapter` elements for your JMS provider as [Example 11–3](#) shows:

**Example 11–3 *jms-adapter* Element With Tibco EMS JMS Configuration**

```
<jms-adapter>
  <name>inboundJmsAdapter1</name>
  ...
  <jndi-provider-url> ... </jndi-provider-url>
  <jndi-factory> ... </jndi-factory>
  <connection-jndi-name> ... </connection-jndi-name>
  <destination-jndi-name> ... </destination-jndi-name>
  ...
</jms-adapter>
```

For all options that the Oracle Event Processing JMS adapters support, see:

- [Section , "JMS Inbound Adapter Component Configuration"](#)
- [Section , "JMS Outbound Adapter Component Configuration"](#)

For specific JMS provider examples, see:

- [Section , "How to Configure a JMS Adapter for Oracle WebLogic Server JMS Manually"](#)
- [Section , "How to Configure a JMS Adapter for Tibco EMS JMS Manually"](#)

For more information, see your JMS service provider documentation.

5. If you specify JMS provider client passwords in the component configuration file, consider encrypting them.

See [Section , "Encrypting Passwords in the JMS Adapter Component Configuration File"](#).

6. Create a JMS client application library that contains the following:
  - The JMS client JAR files your JMS service provider documentation specifies.
  - If you are using Java Object messages, the Java classes used for messaging need to be packaged in a library bundle.

You may include these Java classes in this JMS client JAR application library.

---

**Note:** This JMS client JAR application library must:

- Export all provider-specific packages.
- Export the Java classes used for messaging, if applicable.
- Import `javax.jms` and `javax.naming`.

The application bundle does not need to export the provider-specific packages.

The application bundle must import Java classes used for messaging, if applicable.

---

For more information, see [Section , "Creating Application Libraries"](#).

For a specific JMS provider example, see [Section , "How to Configure a JMS Adapter for Tibco EMS JMS Manually"](#).

7. Copy the JMS client JAR application library to the appropriate Oracle Event Processing server application library directory:
  - a. If your bundle is a driver, you must put it in the library extensions directory. See [Section , "Library Extensions Directory"](#).
  - b. If your bundle is not a driver, you may put it in the library directory. See [Section , "Library Directory"](#)

For more information, see [Section , "How to Update an Application Library Using Oracle Event Processing IDE for Eclipse"](#).

8. If you created a custom converter class in step 3, update the `MANIFEST.MF` file of your application to add the following packages to the `Import-Package` header:

```
Import-Package: javax.jms, javax.naming, ...
...
```

See [Section , "How to Import a Package"](#).

## How to Configure a JMS Adapter for Oracle WebLogic Server JMS Manually

Oracle Event Processing includes a WebLogic JMS client.

When connecting to Oracle WebLogic server, Oracle Event Processing 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 Event Processing server.

For more information, see [Section , "Configuring a JMS Adapter for a JMS Service Provider"](#).

You can manually configure the built-in JMS inbound and outbound adapter to use the Oracle WebLogic Server JMS provider.

The simplest way to create and configure a JMS adapter is using the Oracle Event Processing IDE for Eclipse adapter wizard as [Section , "How to Configure a JMS Adapter Using the Oracle Event Processing IDE for Eclipse"](#) describes. After using the adapter wizard to create and specify the basic JMS adapter configuration, review this procedure to complete the configuration.

**To configure JMS adapters for Oracle WebLogic Server JMS manually:**

1. Update the EPN assembly file of the application by adding a `wlevs:adapter` element for each inbound and outbound JMS adapter you want to use in your application.

[Example 11-4](#) shows the `wlevs:adapter` element for a JMS inbound adapter.

**Example 11-4 `wlevs:adapter` Element for Inbound Adapter**

```
<wlevs:adapter id="inboundJmsAdapter1" provider="jms-inbound">
...
</wlevs:adapter>
```

See:

- [Section , "JMS Inbound Adapter EPN Assembly File Configuration"](#)
  - [Section , "JMS Outbound Adapter EPN Assembly File Configuration"](#)
2. Update the component configuration file of the application by adding a `jms-adapter` element for each inbound and outbound JMS adapter you want to use in your application.

[Example 11-5](#) shows the `jms-adapter` element for the JMS inbound adapter in [Example 11-4](#).

**Example 11-5 `jms-adapter` Element for Inbound Adapter**

```
<jms-adapter>
  <name>inboundJmsAdapter1</name>
...
</jms-adapter>
```

For each `jms-adapter` element, the `name` child element must be set to the corresponding `wlevs:adapter` element `id` child element.

See:

- [Section , "JMS Inbound Adapter Component Configuration"](#)
  - [Section , "JMS Outbound Adapter Component Configuration"](#)
3. Decide how you want to convert between JMS messages and Oracle Event Processing event types:
    - a. If you want the JMS adapters to perform automatic conversion, specify an event type using the `jms-adapter` element `event-type` child element in the JMS adapter component configuration file.



See:

- [Section , "JMS Inbound Adapter Component Configuration"](#)
- [Section , "JMS Outbound Adapter Component Configuration"](#)

- b. If you want the JMS adapters to perform custom conversion, create a custom converter Java class and register it in the EPN assembly file.

See [Section , "Creating a Custom Converter Between JMS Messages and Event Types"](#).

4. Configure the `jms-adapter` elements for your Oracle WebLogic Server JMS provider.

[Example 11–6](#) shows the `jms-adapter` elements for a JMS inbound and JMS outbound adapter.

**Example 11–6 `jms-adapter` Elements for an Oracle WebLogic Server JMS Provider**

```
...
<jms-adapter>
  <name>JmsInbound</name>
  <event-type>SimpleMapEvent</event-type>
  <jndi-provider-url>t3://localhost:7001</jndi-provider-url>
  <destination-jndi-name>QueueIn</destination-jndi-name>
  <user>weblogic</user>
  <password>welcome1</password>
  <work-manager>JettyWorkManager</work-manager>
  <concurrent-consumers>1</concurrent-consumers>
  <session-transacted>>false</session-transacted>
  <delivery-mode>nonpersistent</delivery-mode>
</jms-adapter>

<jms-adapter>
  <name>JmsOutbound</name>
  <event-type>SimpleMapEvent</event-type>
  <jndi-provider-url>t3://localhost:7001</jndi-provider-url>
  <destination-jndi-name>QueueIn</destination-jndi-name>
  <user>weblogic</user>
  <password>welcome1</password>
  <message-selector></message-selector>
  <session-transacted>>false</session-transacted>
  <delivery-mode>nonpersistent</delivery-mode>
</jms-adapter>
...
```

See:

- [Section , "JMS Inbound Adapter Component Configuration"](#)
- [Section , "JMS Outbound Adapter Component Configuration"](#)

5. If you specify JMS provider client passwords in the component configuration file, consider encrypting them.

See [Section , "Encrypting Passwords in the JMS Adapter Component Configuration File"](#).

6. If you are using Java Object messages, you must create a JMS client application library that contains the Java classes used for messaging need to be packaged in a library bundle.

For more information, see [Section , "Creating Application Libraries"](#).

---



---

**Note:** This JMS client JAR application library must:

- Export the Java classes used for messaging.
- Import `javax.jms` and `javax.naming`.

The application bundle does not need to export provider-specific packages.

The application bundle must import Java classes used for messaging, if applicable.

---



---

7. If you are using Java Object messages, copy the Java classes for messaging application library to the appropriate Oracle Event Processing server application library directory:
  - a. If your bundle is a driver, you must put it in the library extensions directory. See [Section , "Library Extensions Directory"](#).
  - b. If your bundle is not a driver, you may put it in the library directory. See [Section , "Library Directory"](#)

For more information, see [Section , "How to Update an Application Library Using Oracle Event Processing IDE for Eclipse"](#).

8. If you created a custom converter class in step 3, update the `MANIFEST.MF` file of your application to add the following packages to the `Import-Package` header:

```
Import-Package: javax.jms, javax.naming, ...
...
```

See [Section , "How to Import a Package"](#).

## How to Configure a JMS Adapter for Tibco EMS JMS Manually

Oracle Event Processing supports TIBCO Enterprise Message Service (EMS) version 4.2.0 or higher.

To use the Tibco EMS JMS provider, you must add the following Tibco EMS client JAR files to the Oracle Event Processing server library directory:

- `tibjms.jar`

For more information, see:

- [Section , "Configuring a JMS Adapter for a JMS Service Provider"](#)
- [Section , "Library Directory"](#)

You can manually configure the built-in JMS inbound and outbound adapter to use the Tibco EMS JMS provider.

The simplest way to create and configure a JMS adapter is using the Oracle Event Processing IDE for Eclipse adapter wizard as [Section , "How to Configure a JMS Adapter Using the Oracle Event Processing IDE for Eclipse"](#) describes. After using the adapter wizard to create and specify the basic JMS adapter configuration, review this procedure to complete the configuration.

### To configure a JMS adapter for Tibco EMS JMS manually:

1. In the EPN assembly file of the application, add a `wlevs:adapter` element for each inbound and outbound JMS adapter you want to use in your application.

[Example 11-7](#) shows the `wlevs:adapter` element for a JMS inbound adapter.

**Example 11-7 `wlevs:adapter` Element for Inbound Adapter**

```
<wlevs:adapter id="inboundJmsAdapter1" provider="jms-inbound">
...
</wlevs:adapter>
```

See:

- [Section , "JMS Inbound Adapter EPN Assembly File Configuration"](#)
  - [Section , "JMS Outbound Adapter EPN Assembly File Configuration"](#)
2. In the component configuration file of the application, add a `jms-adapter` element for each inbound and outbound JMS adapter you want to use in your application.

[Example 11-8](#) shows the `jms-adapter` element for the JMS inbound adapter in [Example 11-7](#).

**Example 11-8 `jms-adapter` Element for Inbound Adapter**

```
<jms-adapter>
  <name>inboundJmsAdapter1</name>
  ...
</jms-adapter>
```

For each `jms-adapter` element, the name child element must be set to the corresponding `wlevs:adapter` element id child element.

See:

- [Section , "JMS Inbound Adapter Component Configuration"](#)
  - [Section , "JMS Outbound Adapter Component Configuration"](#)
3. Decide how you want to convert between JMS messages and Oracle Event Processing event types:
- a. If you want the JMS adapters to perform automatic conversion, specify an event type using the `jms-adapter` element `event-type` child element in the JMS adapter component configuration file.

See:

- [Section , "JMS Inbound Adapter Component Configuration"](#)
- [Section , "JMS Outbound Adapter Component Configuration"](#)

- b. If you want the JMS adapters to perform custom conversion, create a custom converter Java class and register it in the EPN assembly file.

See [Section , "Creating a Custom Converter Between JMS Messages and Event Types"](#).

4. Configure the `jms-adapter` elements for your Tibco EMS JMS provider as [Example 11-9](#) shows:

**Example 11-9 `jms-adapter` Element With Tibco EMS JMS Configuration**

```
<jms-adapter>
  <name>inboundJmsAdapter1</name>
  ...
  <jndi-provider-url>tcp://TIBCOHOST:PORT</jndi-provider-url>
  <jndi-factory>com.tibco.tibjms.naming.TibjmsInitialContextFactory</jndi-factory>
  <connection-jndi-name>CONNECTION_NAME</connection-jndi-name>
  <destination-jndi-name>DESTINATION_NAME</destination-jndi-name>
```

```
...  
</jms-adapter>
```

Where:

- *TIBCOHOST*: the hostname of the Tibco EMS JMS provider host.
- *PORT*: the Tibco EMS JMS provider port.
- *DESTINATION\_NAME*: the destination JNDI name of the Tibco EMS JMS destination, such as `TibcoRequestQueue1`.
- *CONNECTION\_NAME*: the connection JNDI name of the Tibco EMS JMS connection factory you defined in the Tibco EMS JMS server, such as `TibcoQueueConnectionFactory`.

See:

- [Section , "JMS Inbound Adapter Component Configuration"](#)
  - [Section , "JMS Outbound Adapter Component Configuration"](#)
5. If you specify JMS provider client passwords in the component configuration file, consider encrypting them.

See [Section , "Encrypting Passwords in the JMS Adapter Component Configuration File"](#).

6. Create a JMS client application library that contains the following:
- `tibjms.jar`
  - If you are using Java Object messages, the Java classes used for messaging need to be packaged in a library bundle.

You may include these Java classes in this JMS client application library.

---

---

**Note:** The JMS client application library must:

- Export all provider-specific packages.
- Export the Java classes used for messaging, if applicable.
- Import `javax.jms` and `javax.naming`.

The application bundle does not need to export the provider-specific packages.

The application bundle must import Java classes used for messaging, if applicable.

---

---

For more information, see [Section , "Creating Application Libraries"](#).

For a specific JMS provider example, see [Section , "How to Configure a JMS Adapter for Tibco EMS JMS Manually"](#).

7. Copy the application library to the appropriate Oracle Event Processing server application library directory:
- a. If your bundle is a driver, you must put it in the library extensions directory.  
See [Section , "Library Extensions Directory"](#).
  - b. If your bundle is not a driver, you may put it in the library directory.  
See [Section , "Library Directory"](#)

For more information, see [Section , "How to Update an Application Library Using Oracle Event Processing IDE for Eclipse"](#).

8. If you created a custom converter class in step 3, update the `MANIFEST.MF` file of your application to add the following packages to the `Import-Package` header:

```
Import-Package: javax.jms, javax.naming, ...
...
```

See [Section , "How to Import a Package"](#).

## Creating a Custom Converter Between JMS Messages and Event Types

If you want to customize the conversion between JMS messages and event types you must create your own converter bean.

This section describes:

- [Section , "How to Create a Custom Converter for the Inbound JMS Adapter"](#)
- [Section , "How to Create a Custom Converter for the Outbound JMS Adapter"](#)

### How to Create a Custom Converter for the Inbound JMS Adapter

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, JMSEException;
```

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.

See the *Oracle Fusion Middleware Java API Reference for Oracle Event Processing* for a full description of these APIs.

#### To create a custom converter for the inbound JMS adapter:

1. Using the Oracle Event Processing IDE for Eclipse (or your preferred IDE), add a Java class to your application project.
2. Implement the `com.bea.wlevs.adapters.jms.api.InboundMessageConverter` interface.

[Example 11-10](#) shows a possible implementation.

#### **Example 11-10 Custom Converter for an Inbound JMS Adapter**

```
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.JMSEException;
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, JMSEException {
        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, JMSEException {
        TestEvent event = (TestEvent) inputEvent;
        TextMessage message = session.createTextMessage(
            "Text message: " + event.getString_1()
        );
        List<Message> messages = new ArrayList<Message>();
        messages.add(message);
        return messages;
    }
}

```

3. Specify the converter in your application EPN assembly file as [Example 11–11](#) shows:
  - Register the converter class using a bean element.
  - Associate the converter class with the JMS adapter by adding a `wlevs:instance-property` with name set to `converterBean` and `ref` set to the id of bean.

**Example 11–11 Specifying a Converter Class for an Inbound JMS Adapter in the EPN Assembly File**

```

...
<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>
...

```

4. Package the Java class with your application.
 

For more information, see [Chapter 23, "Assembling and Deploying Oracle Event Processing Applications"](#).

## How to Create a Custom Converter for the Outbound JMS Adapter

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, JMSEException;

```

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 Fusion Middleware Java API Reference for Oracle Event Processing* for a full description of these APIs.

**To create a custom converter for the outbound JMS adapter:**

1. Using the Oracle Event Processing IDE for Eclipse (or your preferred IDE), add a Java class to your application project.
2. Implement the `com.bea.wlevs.adapters.jms.api.OutboundMessageConverter` interface.

[Example 11–10](#) shows a possible implementation.

**Example 11–12 Custom Converter for an Outbound JMS Adapter**

```
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;
    }
}
```

3. Specify the converter in your application EPN assembly file as [Example 11–11](#) shows:
  - Register the convert class using a bean element.
  - Associate the converter class with the JMS adapter by adding a `wlevs:instance-property` with name set to `converterBean` and `ref` set to the id of bean.

**Example 11–13 Specifying a Converter Class for an Outbound JMS Adapter in the EPN Assembly File**

```
...
<bean id="myConverter" class="com.customer.MessageConverter"/>
<wlevs:adapter id="jmsOutbound" provider="jms-outbound">
    <wlevs:instance-property name="converterBean" ref="myConverter"/>
</wlevs:adapter>
...
```

4. Package the Java class with your application.

For more information, see [Chapter 23, "Assembling and Deploying Oracle Event Processing Applications"](#).

## Encrypting Passwords in the JMS Adapter Component 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.

---



---

## How to Encrypt Passwords in the JMS Adapter Component Configuration File

You can encrypt the password in the JMS adapter configuration file.

### To encrypt passwords in the JMS adapter component configuration file:

1. Open a command window and set your environment as described in [Section , "Setting Your Development Environment."](#)
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
aesinternal.dat_file
```

where `ORACLE_CEP_HOME` refers to the main BEA directory into which you installed Oracle Event Processing, 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 `aesinternal.dat_file` parameter refers to the location of the `.aesinternal.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>

```

## Configuring the JMS Adapter EPN Assembly File

For each JMS adapter in your event processing network, you must add a corresponding `wlevs:adapter` element to the EPN assembly file of your application; use the `provider` attribute to specify whether the JMS adapter is inbound or outbound.

As with any other stage in the EPN, add listeners to the `wlevs:adapter` element 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 Event Processing application define where in the network the JMS adapters fit in.

For both JMS inbound and outbound 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 element 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` element, 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.

This section describes:

- [Section , "JMS Inbound Adapter EPN Assembly File Configuration"](#)
- [Section , "JMS Outbound Adapter EPN Assembly File Configuration"](#)

For more information, see:

- [Section , "Overview of Component Configuration Files."](#)
- [Section , "Component Configuration Schema wlevs\\_application\\_config.xsd"](#)

## JMS Inbound Adapter EPN Assembly File Configuration

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 destination from which this inbound JMS adapter gets its messages.

Because no converter bean is specified, Oracle Event Processing automatically converts the inbound message to the event type specified in the component configuration file by mapping property names.

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 destination 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 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_1_1_6.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>
```

## JMS Outbound Adapter EPN Assembly File Configuration

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 destination to which this outbound JMS adapter sends messages.

Because no converter bean is specified, Oracle Event Processing automatically converts the incoming event types to outgoing JMS messages by mapping the property names.

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 Event Processing automatically converts these events to JMS messages and sends the messages to the JMS destination configured in the component configuration file associated with the `jmsOutbound` adapter.

```
<?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_1_1_6.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>

```

## Configuring the JMS Adapter Component Configuration File

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.

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.

This section describes:

- [Section , "JMS Inbound Adapter Component Configuration"](#)
- [Section , "JMS Outbound Adapter Component Configuration"](#)

For more information, see:

- [Section , "Overview of Component Configuration Files."](#)
- [Section , "Component Configuration Schema wlevs\\_application\\_config.xsd"](#)

### JMS Inbound Adapter Component Configuration

[Table 11–1](#) lists the `jms-adapter` element child elements applicable to the JMS inbound adapter.

**Table 11–1 jms-adapter Inbound Child Elements**

Child Element	Description
bindings	<p>Bindings are used to configure horizontal scale-out and are an advanced feature. Using the <code>com.oracle.cep.cluster.hagroups.ActiveActiveGroupBean</code>, you can partition an incoming JMS stream in Oracle Event Processing applications by utilizing the notification groups that the <code>ActiveActiveGroupBean</code> creates. Use this element to associate a notification group with a particular <code>message-selector</code> value.</p> <p>For more information, see <a href="#">Section , "ActiveActiveGroupBean"</a></p>
concurrent-consumers	<p>Number of consumers to create. Default value is 1.</p> <p>If you set this value to a number greater than 1:</p> <ul style="list-style-type: none"> <li>■ Consider the <a href="#">work-manager</a> configuration.</li> <li>■ Be sure that your converter bean is thread-safe because the converter bean will be shared among the consumers. For more information, see <a href="#">Section , "Creating a Custom Converter Between JMS Messages and Event Types"</a>.</li> </ul>
connection-jndi-name	<p>Optional. The JNDI name of the JMS connection factory. Default value is <code>weblogic.jms.ConnectionFactory</code>, for Oracle Event Processing server JMS.</p>
connection-password connection-encrypted-password	<p>Optional. Either the password, or encrypted password, for <code>connection-user</code>.</p> <p><b>Note:</b> Specify either <code>connection-password</code> or <code>connection-encrypted-password</code>, but not both.</p> <p>See <a href="#">Section , "Encrypting Passwords in the JMS Adapter Component Configuration File"</a> for details on encrypting the password.</p>
connection-user	<p>Optional. When Oracle Event Processing calls the <code>createConnectionFactory</code> method on the <code>javax.jms.ConnectionFactory</code> to create a connection to the JMS destination (JMS queue or topic), it uses the <code>connection-user</code> and <code>connection-password</code> (or <code>connection-encrypted-password</code>) settings, if configured. Otherwise, Oracle Event Processing uses the user and password (or <code>encrypted-password</code>) settings.</p> <p>You can use the <code>connection-user</code> and <code>connection-password</code> (or <code>connection-encrypted-password</code>) settings in applications where one security provider is used for JNDI access and a separate security provider is used for JMS access.</p>
destination-jndi-name	<p>Required. The JNDI name of the JMS destination.</p> <p><b>Note:</b> Specify either <code>destination-jndi-name</code> or <code>destination-name</code>, but not both.</p>
destination-name	<p>Required. The actual name of the JMS destination.</p> <p><b>Note:</b> Specify either <code>destination-jndi-name</code> or <code>destination-name</code>, but not both.</p>
destination-type	<p>Define the JMS destination type for a JMS adapter. Valid values are <code>TOPIC</code> or <code>QUEUE</code>. This property must be set to <code>TOPIC</code> whenever the <code>durable-subscription</code> property is set to <code>true</code>.</p> <p><b>Note:</b> To support a durable subscription, set this to <code>TOPIC</code> and use it with <code>durable-subscription</code> set to <code>true</code>; give the subscription a unique identifier with <code>durable-subscription-name</code>.</p>
durable-subscription	<p>Specifies whether the JMS topic subscription of a JMS adapter is durable, meaning that it can persist even if subscribers become inactive. Valid values are <code>true</code> or <code>false</code>. This property is only valid if <code>destination-type</code> is set to <code>TOPIC</code>.</p> <p><b>Note:</b> To support a durable subscription, use this with <code>destination-type</code> set to <code>TOPIC</code> and <code>durable-subscription-name</code> set to a unique identifier.</p>

**Table 11–1 (Cont.) jms-adapter Inbound Child Elements**

Child Element	Description
durable-subscription-name	<p>The name to uniquely identify a durable subscription of a JMS adapter. A durable subscription can persist even if subscribers become inactive.</p> <p><b>Note:</b> To support a durable subscription, use this with <code> durable-subscription</code> set to <code>true</code> and <code> destination-type</code> set to <code>TOPIC</code>.</p>
event-type	<p>Event type whose property names match inbound JMS Map Message property names. Specify this property only if you want Oracle Event Processing to automatically perform the conversion between JMS messages and events. If you have created your own custom converter bean, then do not specify this property</p> <p>For more information, see <a href="#">Section , "Creating a Custom Converter Between JMS Messages and Event Types"</a>.</p>
jndi-factory	<p>Optional. The JNDI factory name. Default value is <code>weblogic.jndi.WLInitialContextFactory</code>, for Oracle Event Processing server JMS.</p>
jndi-provider-url	<p>Required. The URL of the JNDI provider.</p>
message-selector	<p>JMS message selector to use to filter messages. Only messages that match the selector will produce events.</p> <p>Default: there is no selector; all messages will produce events.</p>
password encrypted-password	<p>Required. Either the password, or encrypted password, for user.</p> <p><b>Note:</b> Specify either <code>password</code> or <code>encrypted-password</code>, but not both.</p> <p>See <a href="#">Section , "Encrypting Passwords in the JMS Adapter Component Configuration File"</a> for details on encrypting the password.</p>
session-ack-mode-name	<p>Determines how messages are acknowledged. Once a message is successfully acknowledged it will never be resent following a failure.</p> <p>Valid values from <code>javax.jms.Session</code> are:</p> <ul style="list-style-type: none"> <li>▪ <code>AUTO_ACKNOWLEDGE</code>: With this acknowledgment mode, the session automatically acknowledges a client's receipt of a message when the message has been successfully received. Applications that require message redelivery in response to failures during downstream message processing should use the <code>session-transacted</code> property to achieve this.</li> <li>▪ <code>CLIENT_ACKNOWLEDG</code>: With this acknowledgment mode, the client acknowledges a consumed message by calling the message's <code>acknowledge</code> method.</li> <li>▪ <code>DUPS_OK_ACKNOWLEDGE</code>: This acknowledgment mode instructs the session to lazily acknowledge the delivery of messages.</li> </ul> <p>Default: <code>AUTO_ACKNOWLEDGE</code>.</p>
session-transacted	<p>Boolean value that specifies whether or not the session is transactional.</p> <p>If the session is transacted then do not specify <code>session-ack-mode-name</code>.</p> <p>Default: <code>False</code>.</p>
user	<p>Required. When Oracle Event Processing acquires the JNDI <code>InitialContext</code>, it uses the user and password (or <code>encrypted-password</code>) settings.</p>

**Table 11–1 (Cont.) jms-adapter Inbound Child Elements**

Child Element	Description
work-manager	<p>Name of a work manager, configured in the Oracle Event Processing server <code>config.xml</code> file. This name corresponds to the value of the name child element of the <code>work-manager</code> element in <code>config.xml</code>.</p> <p>If <code>concurrent-consumers</code> is greater than 1 and you want all the consumers to be run concurrently, then consider the configuration of the work-manager you associate with this JMS inbound adapter:</p> <ul style="list-style-type: none"> <li>▪ If the <code>work-manager</code> is shared with other components (such as other adapters and Jetty) then set the <code>work-manager</code> attribute <code>max-threads-constraint</code> greater than or equal to the <code>concurrent-consumers</code> setting.</li> <li>▪ If the <code>work-manager</code> is not shared (that is, it is dedicated to this inbound JMS adapter only) then set the <code>work-manager</code> attribute <code>max-threads-constraint</code> equal to the <code>concurrent-consumers</code> setting.</li> </ul> <p>The default value is the work manager configured for the application itself.</p> <p>For more information, see <a href="#">Section , "work-manager"</a>.</p>

The following configuration file shows a complete example of configuring an inbound JMS adapter.

```
<?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>MyWorkManager</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>
```

## JMS Outbound Adapter Component Configuration

[Table 11–2](#) lists the `jms-adapter` element child elements applicable to the JMS outbound adapter.

**Table 11–2 jms-adapter Outbound Component Configuration Child Elements**

Child Element	Description
connection-jndi-name	Optional. The JNDI name of the JMS connection factory. Default value is <code>weblogic.jms.ConnectionFactory</code> , for Oracle Event Processing server JMS.

**Table 11–2 (Cont.) jms-adapter Outbound Component Configuration Child Elements**

Child Element	Description
connection-password connection-encrypted-password	Optional. Either the password, or encrypted password, for connection-user. <b>Note:</b> Specify either connection-password or connection-encrypted-password, but not both. See Section , "Encrypting Passwords in the JMS Adapter Component Configuration File" for details on encrypting the password.
connection-user	Optional. When Oracle Event Processing calls the createConnection method on the javax.jms.ConnectionFactory to create a connection to the JMS destination (JMS queue or topic), it uses the connection-user and connection-password (or connection-encrypted-password) settings, if configured. Otherwise, Oracle Event Processing uses the user and password (or encrypted-password) settings. You can use the connection-user and connection-password (or connection-encrypted-password) settings in applications where one security provider is used for JNDI access and a separate security provider is used for JMS access.
delivery-mode	Specifies the delivery mode: persistent (default value) or nonpersistent.
destination-jndi-name	Required. The JNDI name of the JMS destination. <b>Note:</b> Specify either destination-jndi-name or destination-name, but not both.
destination-name	Required. The actual name of the JMS destination. <b>Note:</b> Specify either destination-jndi-name or destination-name, but not both.
event-type	Event type whose property names match inbound JMS Map Message property names. Specify this property only if you want Oracle Event Processing to automatically perform the conversion between JMS messages and events. If you have created your own custom converter bean, then do not specify this property For more information, see Section , "Creating a Custom Converter Between JMS Messages and Event Types".
jndi-factory	Optional. The JNDI factory name. Default value is weblogic.jndi.WLInitialContextFactory, for Oracle Event Processing server JMS.
jndi-provider-url	Required. The URL of the JNDI provider.
password encrypted-password	Required. Either the password, or encrypted password, for user. <b>Note:</b> Specify either password or encrypted-password, but not both. See Section , "Encrypting Passwords in the JMS Adapter Component Configuration File" for details on encrypting the password.
session-transacted	Boolean value that specifies whether or not the session is transactional. If the session is transacted then do not specify session-ack-mode-name. Default: False.
user	Required. When Oracle Event Processing acquires the JNDI InitialContext, it uses the user and password (or encrypted-password) settings.

The following configuration file shows a complete example of configuring an outbound JMS adapter.

```
<?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>
```



---

---

## Integrating an HTTP Publish-Subscribe Server

This chapter describes how to use the HTTP publish-subscribe server adapter to connect an Oracle Event Processing event processing network with an HTTP pub-sub server.

This chapter includes the following sections:

- [Overview of HTTP Publish-Subscribe Server Adapter Configuration](#)
- [Configuring an HTTP Pub-Sub Adapter](#)
- [Creating a Custom Converter Between the HTTP Pub-Sub Messages and Event Types](#)
- [Configuring the HTTP Pub-Sub Adapter EPN Assembly File](#)
- [Configuring the HTTP Pub-Sub Adapter Component Configuration File](#)

### 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 Event Processing 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.cometd.org/trunk/bayeux/bayeux.html>). The Bayeux protocol defines a contract between the client and the server for communicating with asynchronous messages over HTTP.

In Oracle Event Processing, 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 , "Overview of the Built-In Pub-Sub Adapter for Publishing"](#)
  - Local publishing to a channel: see [Section , "Local Publishing"](#).
  - Remote publishing to a channel: see [Section , "Remote Publishing"](#).
- Subscribing to a channel: see [Section , "Overview of the Built-In Pub-Sub Adapter for Subscribing"](#).

Oracle Event Processing 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 15, "Integrating an External Component Using a Custom Adapter."](#)

By default, Oracle Event Processing performs automatic conversion to and from Oracle Event Processing event types. Alternatively, you can create a custom converter. See [Section , "Creating a Custom Converter Between the HTTP Pub-Sub Messages and Event Types"](#).

Oracle Event Processing can also automatically convert between JSON messages and Oracle Event Processing event types. See [Section , "Converting Between JSON Messages and Event Types"](#).

---

---

**Note:** Byte arrays are not supported as property types in event types used with the pub-sub server.

---

---

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 develop their Web clients using one of the following frameworks:

- Dojo JavaScript library (see <http://dojotoolkit.org/>) that supports the Bayeux protocol. Oracle Event Processing 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 Fusion Middleware Administrator's Guide for Oracle Event Processing*.

## 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:

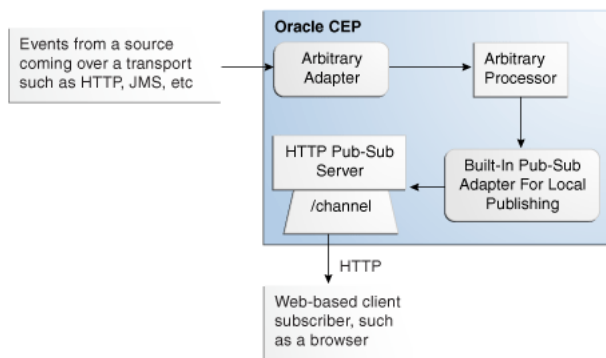
- [Section , "Local Publishing"](#)
- [Section , "Remote Publishing"](#)

For more information, see [Section , "Overview of HTTP Publish-Subscribe Server Adapter Configuration"](#).

### Local Publishing

[Figure 12–1](#) 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 12–1 Built-In Pub-Sub Adapter For Local Publishing**



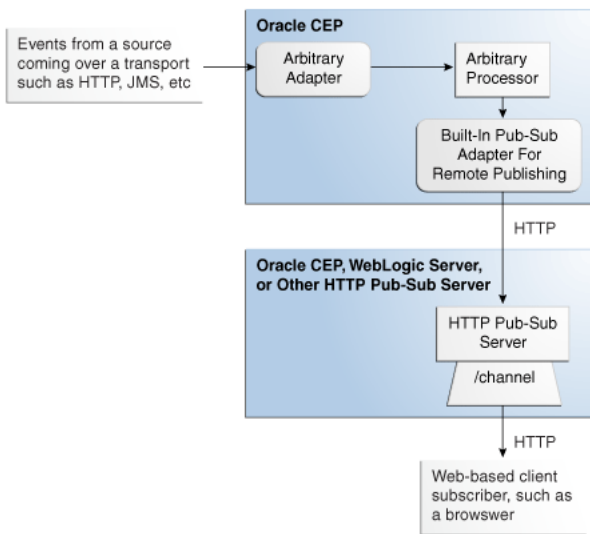
Note the following in [Figure 12–1](#):

- Events flow from some source into an adapter of an application running in Oracle Event Processing. 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 Event Processing 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 Event Processing then sends the event as a message to all subscribers of the local channel.

### Remote Publishing

[Figure 12–2](#) shows how the built-in pub-sub adapter for remote publishing fits into a simple event processing network.

**Figure 12–2 Built-In Pub-Sub Adapter For Remote Publishing**



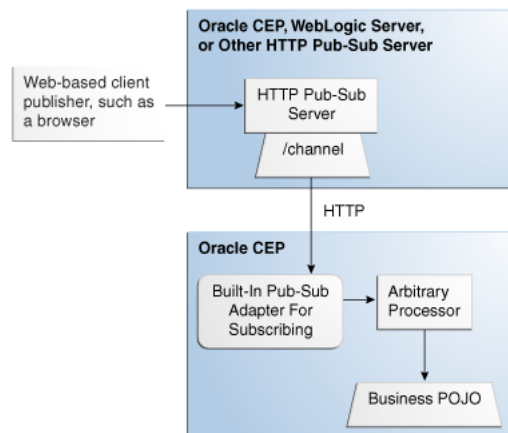
Note the following in [Figure 12–2](#):

- Events flow from some source into an adapter of an application running in Oracle Event Processing. 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 Event Processing 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.

## Overview of the Built-In Pub-Sub Adapter for Subscribing

Figure 12-3 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 12-3 Built-In Pub-Sub Adapter For Subscribing**



Note the following in Figure 12-3:

- Messages are published to a remote HTTP pub-sub server, which could be another instance of Oracle Event Processing, 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 Event Processing 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 Event Processing application.

- The processor sends the events to a business POJO. Again, this business POJO is not required.

For more information, see [Section , "Overview of HTTP Publish-Subscribe Server Adapter Configuration"](#).

## Converting Between JSON Messages and Event Types

Oracle Event Processing can automatically convert incoming JavaScript Object Notation (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 , "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 Event Processing take care of the conversion between messages and event types, the following is true:

- You must specify an event type that Oracle Event Processing uses in its conversion. See [Section , "How to Configure an HTTP Pub-Sub Adapter Manually"](#) 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.

For more information, see <http://www.json.org/>.

## Configuring an HTTP Pub-Sub Adapter

This section describes how to configure Oracle Event Processing HTTP pub-sub adapter for both publishing and subscribing:

- [Section , "How to Configure an HTTP Pub-Sub Adapter Using the Oracle Event Processing IDE for Eclipse"](#)
- [Section , "How to Configure an HTTP Pub-Sub Adapter Manually"](#)

## How to Configure an HTTP Pub-Sub Adapter Using the Oracle Event Processing IDE for Eclipse

The simplest way to create and configure an HTTP pub-sub adapter is using the Oracle Event Processing IDE for Eclipse adapter wizard.

For more information, see [Section , "How to Create an Adapter Node"](#).

After using the adapter wizard to create and specify the basic HTTP pub-sub adapter configuration, review [Section , "How to Configure an HTTP Pub-Sub Adapter Manually"](#) to complete the configuration.

## How to Configure an HTTP Pub-Sub Adapter Manually

This section describes how to create and configure an HTTP pub-sub adapter manually. It describes the detailed steps that you may require depending on your application.

The simplest way to create and configure an HTTP pub-sub adapter is using the Oracle Event Processing IDE for Eclipse adapter wizard as [Section , "How to Configure an HTTP Pub-Sub Adapter Using the Oracle Event Processing IDE for Eclipse"](#) describes. After using the adapter wizard to create and specify the basic HTTP pub-sub adapter configuration, review this procedure to complete the configuration.

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 , "Overview of Component Configuration Files."](#)

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 , "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 manually:

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 Event Processing 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 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>

```

- 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 Event Processing application will publish or subscribe, respectively. The remote pub-sub server could be another instance of Oracle Event Processing, 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`.

- 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 Event Processing instance hosting the current Oracle Event Processing application.

By default, each Oracle Event Processing 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>

```

- 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`.

- 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 Event Processing 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 Event Processing 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>

```

8. 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 Event Processing make the conversion based on mapping property names between the messages and a specified event type.

See [Section , "Creating a Custom Converter Between the HTTP Pub-Sub Messages and Event Types."](#)

9. If you are going to use the local HTTP pub-sub server associated with the Oracle Event Processing instance for local publishing, use Visualizer, the Oracle Event Processing 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 Fusion Middleware Visualizer User's Guide for Oracle Event Processing*.

10. Update the EPN assembly file, adding declarations for each built-in pub-sub adapter you are adding to your application.

See [Section , "Configuring the HTTP Pub-Sub Adapter EPN Assembly File."](#)

11. 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="11.1.1.4_0",
  ...

```

See [Section , "Creating the MANIFEST.MF File"](#) for additional information on the manifest file.



## 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 Fusion Middleware Java API Reference for Oracle Event Processing* 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;
    }
}
```

You can use the GSON Java library to help you convert Java objects to JSON format.

For more information, see:

- <http://www.json.org/>

- <http://code.google.com/p/google-gson>

## Configuring the HTTP Pub-Sub Adapter EPN Assembly File

For each HTTP pub-sub adapter in your event processing network, you must add a corresponding `wlevs:adapter` element to the EPN assembly file of your application; use the `provider` attribute to specify whether the HTTP pub-sub adapter is publishing or subscribing.

As with any other stage in the EPN, add listeners to the `wlevs:adapter` element to integrate the HTTP pub-sub adapter into the event processing network. The requirements of your own Oracle Event Processing application define where in the EPN the HTTP pub-sub adapters fit in.

This section describes:

- [Section , "HTTP Pub-Sub Adapter for Publishing EPN Assembly File Configuration"](#)
- [Section , "HTTP Pub-Sub Adapter for Subscribing EPN Assembly File Configuration"](#)

For more information, see:

- [Section , "Overview of Component Configuration Files."](#)
- [Section , "Component Configuration Schema `wlevs\_application\_config.xsd`"](#)

## HTTP Pub-Sub Adapter for Publishing EPN Assembly File Configuration

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.

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 Event Processing 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 , "How to Configure an HTTP Pub-Sub Adapter Manually"](#)), `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 Event Processing 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 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 , "How to Configure an HTTP Pub-Sub Adapter Manually"](#)), `remoteSubscriber` subscribes to a channel called `/channel3` configured for the remote HTTP pub-sub server hosted at `http://myhost.com:9102/pubsub`. Oracle Event Processing 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.

## HTTP Pub-Sub Adapter for Subscribing EPN Assembly File Configuration

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.

The value of the `id` attribute, in this case `remoteSubscriber`, must match the name specified for this HTTP pub-sub adapter in its configuration file. The configuration file configures the HTTP pub-sub server destination to which this adapter subscribes.

The following sample EPN file shows an event processing network with one built-in pub-sub adapter for subscribing:

```

<?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>

```

## Configuring the HTTP Pub-Sub Adapter Component Configuration File

You configure the HTTP 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.

The root element for configuring an HTTP pub-sub adapter is `http-pub-sub-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.

This section describes:

- [Section , "HTTP Pub-Sub Adapter for Publishing Component Configuration"](#)
- [Section , "HTTP Pub-Sub Adapter for Subscribing Component Configuration"](#)

For more information, see:

- [Section , "Overview of Component Configuration Files."](#)
- [Section , "Component Configuration Schema wlevs\\_application\\_config.xsd"](#)

### HTTP Pub-Sub Adapter for Publishing Component Configuration

[Table 12–1](#) lists the `http-pub-sub-adapter` element child elements applicable to an HTTP pub-sub adapter for publishing.

**Table 12–1 http-pub-sub-adapter for Publishing Component Configuration Child Elements**

Child Element	Description
server-context-path	<p>Required. For each <i>local</i> HTTP pub-sub adapter for publishing, specify the value of the Oracle Event Processing server <code>config.xml</code> file element <code>http-pubsub</code> child element <code>path</code> of the local HTTP pub-sub server associated with the Oracle Event Processing instance hosting the current Oracle Event Processing application.</p> <p>Default: <code>/pubsub</code>.</p> <p>If you have created a new local HTTP pub-sub server, or changed the default configuration, then specify the appropriate <code>path</code> child element value.</p> <p><b>NOTE:</b> Do not specify this option for a remote HTTP pub-sub adapter.</p>
server-url	<p>Required. For each <i>remote</i> HTTP pub-sub adapter for publishing, specify the URL of the remote HTTP pub-sub server to which the Oracle Event Processing application will publish. The remote HTTP pub-sub server could be another instance of Oracle Event Processing, or a WebLogic Server instance, or it could be any third-party HTTP pub-sub server. For example:</p> <p><code>http://myhost.com:9102/pubsub</code></p> <p><b>NOTE:</b> Do not specify this option for a local HTTP pub-sub adapter.</p>
channel	<p>Required. For both local and remote HTTP pub-sub adapters for publishing, specify the channel that the HTTP pub-sub adapter publishes to.</p>
event-type	<p>Optional. For both local and remote HTTP pub-sub adapters for publishing, specify the fully qualified class name of the <code>JavaBean</code> event to limit the types of events that are published. Otherwise, all events sent to the HTTP pub-sub adapter are published.</p> <p>You must register this class in the EPN assembly file as a <code>wlevs:event-type-repository</code> element <code>wlevs:class</code> child element. For more information, see <a href="#">Section , "Creating an Oracle Event Processing Event Type as a JavaBean"</a>.</p>
user	<p>Optional. For both local and remote HTTP pub-sub adapters for publishing, if the HTTP pub-sub server to which the Oracle Event Processing application is publishing requires user authentication, specify a user name.</p>
password	<p>Optional. For both local and remote HTTP pub-sub adapters for publishing, if the HTTP pub-sub server to which the Oracle Event Processing application is publishing requires user authentication, specify a password.</p> <p>Choose either <code>password</code> or <code>encrypted-password</code> but not both.</p>
encrypted-password	<p>Optional. For both local and remote HTTP pub-sub adapters for publishing, if the HTTP pub-sub server to which the Oracle Event Processing application is publishing requires user authentication and requires password encryption, specify an encrypted password.</p> <p>Choose either <code>password</code> or <code>encrypted-password</code> but not both.</p>

The following configuration file shows a complete example of configuring an HTTP pub-sub adapter for publishing: both a remote and local publisher is shown.

```
<?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>
</nl:config>

```

## HTTP Pub-Sub Adapter for Subscribing Component Configuration

Table 12–2 lists the `http-pub-sub-adapter` element child elements applicable to an HTTP pub-sub adapter for subscribing.

**Table 12–2** *http-pub-sub-adapter for Subscribing Component Configuration Child Elements*

Child Element	Description
<code>server-url</code>	Required. For each <i>remote</i> HTTP pub-sub adapter for subscribing, specify the URL of the remote HTTP pub-sub server to which the Oracle Event Processing application will publish. The remote HTTP pub-sub server could be another instance of Oracle Event Processing, or a WebLogic Server instance, or it could be any third-party HTTP pub-sub server <b>NOTE:</b> do not specify this option for a local HTTP pub-sub adapter.
<code>channel</code>	Required. For both local and remote HTTP pub-sub adapters for subscribing, specify the channel that the HTTP pub-sub adapter subscribes to.
<code>event-type</code>	Required. For both local and remote HTTP pub-sub adapters for subscribing, specify the fully qualified class name of the <code>JavaBean</code> to which incoming messages are mapped. At runtime, Oracle Event Processing uses the incoming key-value pairs in the message to map the message data to the specified event type.  You must register this class in the EPN assembly file as a <code>wlevs:event-type-repository</code> element <code>wlevs:class</code> child element. For more information, see Section , "Creating an Oracle Event Processing Event Type as a <code>JavaBean</code> ".
<code>user</code>	Optional. For both local and remote HTTP pub-sub adapters for subscribing, if the HTTP pub-sub server to which the Oracle Event Processing application is publishing requires user authentication, specify a user name.
<code>password</code>	Optional. For both local and remote HTTP pub-sub adapters for subscribing, if the HTTP pub-sub server to which the Oracle Event Processing application is publishing requires user authentication, specify a password. Choose either <code>password</code> or <code>encrypted-password</code> but not both.
<code>encrypted-password</code>	Optional. For both local and remote HTTP pub-sub adapters for subscribing, if the HTTP pub-sub server to which the Oracle Event Processing application is publishing requires user authentication and requires password encryption, specify an encrypted password. Choose either <code>password</code> or <code>encrypted-password</code> but not both.

The following configuration file shows a complete example of configuring an HTTP pub-sub adapter for subscribing.

```

<?xml version="1.0" encoding="UTF-8"?>
<nl:config
  xsi:schemaLocation="http://www.bea.com/ns/wlevs/config/application wlevs_application_
config.xsd"
  xmlns:nl="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>
<b>http-pub-sub-adapter</b>
  <b>name>remoteSubscriber</b></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>
</nl:config>
```



---

---

## Integrating a Cache

This chapter describes how to configure a caching system for use with Oracle Event Processing event processing networks. It includes information on how to configure caches that are based on Oracle Coherence, Oracle Event Processing, and third-party caching providers and to access them from Oracle Continuous Query Language, Java classes, and other code.

This chapter includes the following sections:

- [Overview of Integrating a Cache](#)
- [Configuring an Oracle Coherence Caching System and Cache](#)
- [Configuring an Oracle Event Processing Local Caching System and Cache](#)
- [Configuring a Third-Party Caching System and Cache](#)
- [Adding Caching to an Event Processing Network](#)
- [Accessing a Cache from Application Code](#)

### Overview of Integrating a Cache

You can integrate a cache system with your Oracle Event Processing application so that the cache is available as source or destination for data your application uses, including event data. Integrating a cache can provide access to relatively static data at a speed that is well suited to an application that handles streaming data.

A *cache* is a temporary storage area for events, created to improve the overall performance of your Oracle Event Processing application (a cache is not necessary for the application to function correctly). To increase the availability of the events and increase the performance of their applications, Oracle Event Processing applications can publish to or consume events from a cache.

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.

By integrating a cache, you can:

- Load into the cache frequently-used data for access from application code, improving application performance.
- Write to the cache processed event data for use by another application.

For more complete cache integration uses case descriptions, see [Section , "Caching Use Cases"](#).

Note that a cache 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.

Oracle Event Processing 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.

Although configuration steps vary for each of the caching implementations, the high-level steps are as follows:

**To integrate caching:**

1. Configure the caching system and caches. How you do this will vary depending on the caching implementation you're using.  
For more information, see [Section , "Overview of Cache Configuration"](#).
2. Declare the caching system and its caches by updating a component configuration file.  
For more information, see [Section , "Overview of Cache Configuration"](#).
3. Add the caching system and caches to the event processing network by editing the EPN assembly file. Note that you can use the IDE's EPN editor to add a cache.  
For more information, see [Section , "Adding Caching to an Event Processing Network"](#).
4. Write code to access the cache in your Oracle Event Processing application.  
For more information, see [Section , "Accessing a Cache from Application Code"](#).

## Caching Implementations Supported by Oracle Event Processing

Oracle Event Processing supports the following caching implementations:

- Oracle Event Processing local cache: a local, in-memory single-JVM cache. This implementation is best for local use (it cannot be used in a cluster). It might also be useful for development in the early stages because it is relatively simple to set up.
- 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 Event Processing 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 Event Processing to work with other, third-party cache implementations.

## Overview of Cache Configuration

You configure caching systems and caches before adding them to the event processing network (EPN). How you add configuration depends on which caching implementation you will be using.

You integrate caching systems and caches by updating the following configuration files:

- If you are using Oracle Coherence for caching, use a `coherence-cache-config.xml` file to configure it. This is the same kind of file you use to configure Coherence in other contexts.

For more information, see [Section , "Configuring the Oracle Coherence Caching System and Caches"](#).

- If you are using Oracle Coherence, you can optionally use a `tangosol-coherence-override.xml` file to make global, server-wide configuration changes if you are caching with Oracle Coherence.

For more information, see [Section , "Configuring the Oracle Coherence Caching System and Caches"](#).

- Use a component configuration file for basic configuration of caching systems and caches. In this file, you essentially declare the existence of the cache and provide configuration for it. If the cache is an Oracle Coherence cache, you can use this file to reference Coherence-specific configuration in a `coherence-cache-config.xml` file.

For each caching system your application uses, you add a separate `caching-system` element in a component configuration file. You also add a cache element for each cache.

For more information, see [Section , "Configuring the Oracle Coherence Caching System and Caches"](#) or [Section , "Configuring an Oracle Event Processing Caching System"](#).

- Use the application's EPN assembly file to add configured caching systems and caches to the event processing network, connecting the caches to other components in the EPN.

For more information, see [Section , "Adding Caching to an Event Processing Network"](#).

Basically, though, you create, configure, and wire caching systems and caches using component configuration and EPN assembly files. By default, when you create a project, the IDE creates one component configuration file and one EPN assembly file. When you add a cache to the EPN using the IDE, it adds a cache element to the EPN

assembly file. You must manually add `caching-system` elements to the EPN assembly and component configuration files.

When adding configured caching systems and caches to an application's event processing network, you specify entries in the EPN assembly file with entries in the component configuration file by referencing configured names.

In [Example 13-1](#) (a component configuration file excerpt) and [Example 13-2](#) (an EPN assembly file excerpt), EPN `id` attribute values must have the same values as configuration name attribute values:

**Example 13-1 Component Configuration File Name Values**

```
<caching-system>
  <name>cacheSystem</name>
  <cache>
    <name>cache1</name>
    ...
  </cache>
</caching-system>
```

**Example 13-2 EPN Assembly File ID and Ref Values**

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

<wlevs:cache id="cache1">
  <wlevs:caching-system ref="cacheSystem" />
</wlevs:cache>
```

If your application has more than one caching system, you can create a `caching-system` element for each of them in a `config.xml` file. You can use one XML file or several, putting all in the `META-INF/wlevs` directory. Choose the method that best suits your development environment.

For more information, see:

- [Section , "Overview of Component Configuration Files"](#)
- [Section , "Overview of EPN Assembly Files"](#)
- [Section , "Creating EPN Assembly Files"](#)
- *Oracle Fusion Middleware Visualizer User's Guide for Oracle Event Processing*
- "wlevs.Admin Command-Line Reference" in the *Oracle Fusion Middleware Administrator's Guide for Oracle Event Processing*
- "Configuring JMX for Oracle Event Processing" in the *Oracle Fusion Middleware Administrator's Guide for Oracle Event Processing*

For more information on configuring caching, see:

- [Section , "Caching Use Cases"](#)
- [Section , "Caching APIs"](#)

## Caching Use Cases

Caching technology is a great fit for streaming data use cases, where high throughput can be particularly important. Getting data from a cache will usually be much faster than getting the same data from a relational database.

The following describes common use cases for caching in Oracle Event Processing applications.

- **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 Event Processing 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 Event Processing application that generates events (input adapter, channel, business POJO, or processor) to publish its events to the cache.

- **Consuming data from a cache**

Oracle Event Processing 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 Event Processing 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 , "Configuring Oracle Event Processing Resource Access"](#).

Applications can also query a cache directly from an Oracle CQL or EPL statement that runs in a processor. In this case, the cache essentially functions as another type of 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.

- **Updating and deleting data in a cache**

An Oracle Event Processing 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.

- **Using a cache in a multi-server domain**

If you build an Oracle Event Processing 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:

- "Administering Multi-Server Domains With Oracle Event Processing Native Clustering" in the *Oracle Fusion Middleware Administrator's Guide for Oracle Event Processing*
- [Section , "Configuring an Oracle Coherence Caching System and Cache"](#)
- [Section , "Configuring a Third-Party Caching System and Cache"](#)

## Caching APIs

Oracle Event Processing provides caching APIs that you can use in your application to perform certain tasks. The APIs are in the `com.bea.cache.jcache` package, which includes the APIs used to access a cache and create cache loader, listeners, and stores. Also, if you intend to use that functionality, you will need to import the `com.tangosol.net` and `com.tangosol.net.cache` packages.

You create, configure, and wire caching systems and caches 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 Event Processing local cache, then the 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 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 beans you write must implement the appropriate third-party cache interfaces.

For more information, see:

- [Section , "Configuring an Oracle Coherence Caching System and Cache"](#)
- [Section , "Configuring an Oracle Event Processing Local Caching System and Cache"](#)
- [Section , "Configuring a Third-Party Caching System and Cache"](#)
- *Oracle Fusion Middleware Java API Reference for Oracle Event Processing*.

## 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 Event Processing will ignore any other caching systems you might configure.

---



---

**Note:** Before you can legally use Oracle Event Processing 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>.

---



---

### To configure an Oracle Coherence caching system and cache:

1. Decide whether or not your Oracle Coherence cache will be used exclusively by this application or shared amongst two or more applications.

For more information, see [Section , "Configuring a Shared Oracle Coherence Cache"](#).

2. Configure the cache and caching system using a `coherence-cache-config.xml` file (and possibly a `tangosol-coherence-override.xml` file) and place the file in your application's `META-INF/wlevs/coherence` directory.

3. Configure the caching system and its caches by updating the caching configuration file for the application.

See [Section , "Configuring the Oracle Coherence Caching System and Caches."](#)

4. Configure the caching system and its caches by updating the EPN assembly file with one or more cache element child elements.

5. Before assembling and deploying the application, edit your `META-INF/MANIFEST.MF` to import packages that might be required in your implementation. If your application implements cache listeners, loaders or stores, your manifest should import `com.tangosol.net.cache` packages.

For more information, see [Section , "How to Import a Package"](#).

## Configuring the Oracle Coherence Caching System and Caches

When configuring an Oracle Coherence cache for integration with an Oracle Event Processing application, you use Coherence configuration files, then reference those files in Oracle Event Processing component configuration.

Oracle Event Processing leverages the native configuration provided by Oracle Coherence. You do this by packaging the following two Oracle Coherence configuration files, using the file names indicated in the following list, 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 , "The coherence-cache-config.xml File"](#) for information about this file as well as an example of the mapping.

This 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 Event Processing caching provider (`META-INF/wlevs`).

- `tangosol-coherence-override.xml`—Oracle Coherence cluster configuration. See [Section , "The tangosol-coherence-override.xml File"](#) for information about this file as well as an example.

This is a global per-server file (referred to as "operational configuration" in the Oracle Coherence documentation); put this file in the Oracle Event Processing server config directory.

Once you have configured Oracle Coherence using the files in the preceding list, update your component configuration file as shown in [Example 13-3](#). Here, you declare a Coherence caching system for use in the application by referencing the `coherence-cache-config.xml` file where you configured the cache.

### **Example 13-3 Component 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 Event Processing local configuration. Otherwise, Oracle Event Processing will throw an exception. For reference information on this file, see [Section , "coherence-caching-system"](#) and [Section , "coherence-cache-config"](#).

### **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>
<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>
<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 Event Processing: 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 Event

Processing 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.

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.

### The `tangosol-coherence-override.xml` File

The `tangosol-coherence-override.xml` file configures Oracle Coherence caching. Include 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 Event Processing starts up; this can cause problems and sometimes even prevent Oracle Event Processing 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 Event Processing clusters, see "Administrating Multi-Server Domains With Oracle Event Processing Native Clustering" in the *Oracle Fusion Middleware Administrator's Guide for Oracle Event Processing*.

## Configuring a Shared Oracle Coherence Cache

When declaring Oracle Coherence caches in the EPN assembly files of one or more applications deployed to the same Oracle Event Processing 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 Event Processing 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`.

To export both the caching system and the cache as an OSGi service, set the `advertise` attribute to `true`.

```
<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>
```

## Configuring an Oracle Event Processing Local Caching System and Cache

You can configure your application to use the Oracle Event Processing local caching system and cache. The Oracle Event Processing 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 , "Configuring an Oracle Coherence Caching System and Cache"](#)).

### To configure an Oracle Event Processing local caching system and cache:

1. Declare the caching system and caches by updating a component configuration file by adding `caching-system` and `cache` elements.

For more information, see [Section , "Configuring an Oracle Event Processing Caching System"](#)

2. Optionally, override the default cache configuration by updating the EPN assembly file with one or more additional `cache` element child elements.

For more information, see [Section , "Adding Caching to an Event Processing Network"](#).

3. Before assembling and deploying the application, verify that the `META-INF/MANIFEST.MF` file includes the following import:

```
com.bea.wlevs.cache.spi; version = "11.1.0.0"
```

For more information, see [Section , "How to Import a Package"](#).

## Configuring an Oracle Event Processing Caching System

In order to use an Oracle Event Processing caching system cache in an application, you need to first configure the cache. Afterward, you can reference the cache in other places, such as an event processing network. This section describes the settings you can make in a component configuration file. For reference information on the file, see [Section , "caching-system"](#) and [Section , "cache"](#).

In a component configuration file (such as a `config.xml` file), to the `config` root element, add a `caching-system` child element; use its `name` child element to uniquely identify the caching system. This name will match the EPN assembly file's `wlevs:caching-system` element `id` attribute. This is how Oracle Event Processing 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 from this example 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>
```

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 each.

This name must match the `wlevs:cache` element `id` attribute you specify in the EPN assembly file. This is how Oracle Event Processing knows to which particular cache in the EPN assembly file this configuration applies.

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 more information, see [Section , "work-manager"](#).

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>
```

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 (`<write-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. For more information, see [Section , "work-manager"](#).
- `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:

```
<キャッシング-system>
  <name>キャッシング-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>
</キャッシング-system>
```

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 Event Processing server `config.xml` configuration file.

For example:

```
<キャッシング-system>
  <name>キャッシング-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>

```

For more information, see [Section , "work-manager"](#)

## Configuring a Third-Party Caching System and Cache

You can configure your application to use a third-party caching system and cache.

### To configure a third-party caching system and cache:

1. Create a plug-in to define the third-party caching system as an Oracle Event Processing 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` attribute to specify additional information.

For simplicity, you can include the third-party implementation code inside the Oracle Event Processing 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 Fusion Middleware Java API Reference for Oracle Event Processing*.

Sometimes, however, you might not be able, or want, to include the third-party caching implementation in the same bundle as the Oracle Event Processing 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. When you assemble your application, verify that the `META-INF/MANIFEST.MF` file includes the following import:

```
com.bea.wlevs.cache.spi; version = "11.1.0.0"
```

If the `MANIFEST.MF` file does not include this import, update the `MANIFEST.MF` file to add this import before deploying your application.

## Adding Caching to an Event Processing Network

Once you have configured the caching provider your application will be using, you can add it to the application's event processing network (EPN).

For information about adding caching to the EPN and configuring it to be reused, see the following sections:

- [Section , "Adding the Caching System and Caches to an EPN"](#)
- [Section , "Configuring a Cache for Reuse Among Applications"](#)

When you add a cache to the EPN, you can specify other functionality, as described in the following sections:

- [Section , "Configuring a Cache as an Event Listener"](#)
- [Section , "Configuring a Cache as an Event Source"](#)
- [Section , "Exchanging Data Between a Cache and Another Data Source"](#)

## Adding the Caching System and Caches to an EPN

To declare a caching system that uses the Oracle Coherence implementation declaratively in the EPN assembly file, use the `wlevs:caching-system` element, whose `id` attribute must match the name you specified for the caching system in the application's component configuration file.

After you have declared a caching system for an application, you configure one or more caches using the `wlevs:cache` element. The element's mandatory `id` attribute maps to the name of a cache in the configuration file.

The following example illustrates a caching system and cache declared in an EPN assembly file:

```
<wlevs:caching-system id="caching-system-id" provider="coherence" advertise="false"/>
...
<wlevs:cache id="myCache" advertise="false">
  <wlevs:caching-system ref="caching-system-id"/>
</wlevs:cache>
```



```
</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.

## Configuring a Cache for Reuse Among Applications

You can export both the caching system and the cache as OSGI services using the `advertise` attribute. 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 illustrates 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>
```

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>
```

For more information, see [Section , "Configuring a Shared Oracle Coherence Cache"](#).

## Configuring a Cache as an Event Listener

You can configure a cache in an EPN to receive events as they pass through the network.

For example, to specify that a cache listens to a channel, configure the channel with a `wlevs:listener` element that has a reference to the cache, 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: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 the channel sends a *remove event* (an old event that exits the output window), then the event is removed from the cache.

## Specifying the Key Used to Index a Cache

When you configure a 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 , "Specifying a Key Property in EPN Assembly File"](#)
- [Section , "Using a Metadata Annotation to Specify a Key"](#)
- [Section , "Specifying a Composite Key"](#)

**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 Event Processing 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>
```

**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.

**To use a metadata annotation to specify a key:**

1. Import the `com.bea.wlevs.ede.api.Key` package.

For more information, see [Section , "How to Import a Package"](#).

2. Apply the `@Key` annotation to a method.

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 {
    private String key;
```

```

public MyEvent() {
}
public MyEvent(String key) {
    this.key = key;
}
public String getKey() {
    return key;
}
@Key
public void setKey(String key) {
    this.key = key;
}
}

```

**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>

```

## Configuring a Cache as an Event Source

You can configure a 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 bean.

A class that listens to a cache must implement an interface that provides methods for receiving events, as follows:

- A class that listens to a Coherence cache must implement the `com.tangosol.util.MapListener` interface.
- A class that listens to an Oracle Event Processing local cache must implement the `com.bea.cache.jcache.CacheListener` interface.

```

<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="cacheListenerId" class="wlevs.example.LocalListener"/>

```

In the example, the `cacheListenerId` Spring bean listens to events coming from the cache. In this case, the class that implements this component, `com.bea.wlevs.example.MyCacheListener`, is listening to an Oracle Coherence cache. It must implement the appropriate Oracle Coherence-specific Java interfaces, including `com.tangosol.util.MapListener`. [Example 13-4](#) illustrates this implementation.

### **Example 13-4 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++;
    }
}
```

## Exchanging Data Between a Cache and Another Data Source

You can have a cache in an EPN exchange data with another data source, including a database. For example, you can load a cache with data when the application starts or create a read/write relationship between the cache and a database.

If the cache will only be reading data, including when the backing store is read-only, you should use a cache loader. If the cache will read and write data, use a cache store. In both cases, creating the relationship involves specific configuration and a Java class that knows how to communicate with the data source.

For more information, see the following topics:

- [Section , "Loading Cache Data from a Read-Only Data Source"](#)
- [Section , "Exchanging Data with a Read-Write Data Source"](#)

### Loading Cache Data from a Read-Only Data Source

Using a cache loader, you can have a cache in your EPN load data from a read-only data source. A cache loader is a Java class that loads cache objects into a cache. You create a cache loader by writing a Java class that implements the appropriate interfaces to enable the loader class to communicate with the cache. Then 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.

If the backing store is read-write, use a cache store instead (see [Section , "Exchanging Data with a Read-Write Data Source"](#)).

When creating a cache loader, you implement interfaces as follows:

- To load cache data into an Oracle Coherence cache, create a class that implements the appropriate Oracle Coherence-specific Java interfaces, including `com.tangosol.net.cache.CacheLoader`. See [Example 13-6](#) for an example.
- To load cache data into an Oracle Event Processing local cache, create a class that implements `com.bea.cache.jcache.CacheLoader` interface. This interface includes the `load` method to customize loading a single object into the cache; Oracle Event Processing 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.

In [Example 13-5](#), the `localLoader` bean loads events into an Oracle Coherence cache when the backing store is read-only.

When working with a Coherence cache, note that if you specify a cache loader in your configuration file, you must also specify the corresponding class factory method name

in your Coherence cache configuration file. For a cache loader, you specify the `getLoader` method of `com.bea.wlevs.cache.coherence.configuration.SpringFactory`. For example code, see [Section , "The coherence-cache-config.xml File"](#).

#### **Example 13–5 Oracle Coherence Cache EPN Assembly File for a Cache Loader**

```
<wlevs: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 13–6 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;
    }
}
```

### **Exchanging Data with a Read-Write Data Source**

Using a cache store, you can have a cache in your EPN exchange data with a read-write data source. A cache store is a Java class that exchanges cache objects with a cache. You create a cache store by writing a Java class that implements the appropriate interfaces to enable the it to communicate with the data source. Then you add the cache store to the EPN by using the `wlevs:cache-loader` child element of the `wlevs:cache` element to specify the bean that communicates with the data source.

If the backing store is read-only, use a cache loader instead (see [Section , "Loading Cache Data from a Read-Only Data Source"](#)).

When creating a cache store, you implement interfaces as follows:

- To exchange cache data with an Oracle Coherence cache, create a class that implements the appropriate Oracle Coherence-specific Java interfaces, including `com.tangosol.net.cache.CacheStore`. See [Example 13-8](#) for an example.
- To exchange cache data with an Oracle Event Processing local cache, create a class that implements 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 Event Processing 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.

In [Example 13-7](#), the `localStore` bean loads events into the cache when the backing store is read-write.

Note that if you specify a cache store in your Spring configuration file, you must also specify the corresponding class factory method name in your Coherence cache configuration file. For a cache store, you specify the `getStore` method of `com.bea.wlevs.cache.coherence.configuration.SpringFactory`. For example code, see [Section , "The coherence-cache-config.xml File"](#).

#### **Example 13-7 Oracle Coherence Cache EPN Assembly File for a Cache Store**

```
<wlevs: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 13-8 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;
}
}
}

```

## Accessing a Cache from Application Code

Once you have configured a cache, you can access the cache from several components in an Oracle Event Processing application. This section describes how to do that.

For more information, see the following sections:

- [Section , "Accessing a Cache from an Oracle CQL Statement"](#)
- [Section , "Accessing a Cache From an EPL Statement"](#)
- [Section , "Accessing a Cache from an Adapter"](#)
- [Section , "Accessing a Cache From a Business POJO"](#)
- [Section , "Accessing a Cache From an Oracle CQL User-Defined Function"](#)
- [Section , "Accessing a Cache From an EPL User-Defined Function"](#)
- [Section , "Accessing a Cache Using JMX"](#)

## 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 an event source such as a channel; this feature enables you to enrich standard streaming data with data from a separate source. [Example 13–9](#) 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 13–9 Valid Oracle CQL Query Against a Cache**

```

SELECT S1.symbol, S1.lastPrice, stockCache.description
FROM S1 [Now], stockCache
WHERE S1.symbol = stockCache.symbol

```

You must abide by these restrictions when using a cache in an Oracle CQL query:

- 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.

The following example 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 Event Processing server error "external relation must be joined with s[now]".

```
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 Event Processing *pulls* the data rather than it being *pushed*, as is the case with a channel. This means that, continuing with [Example 13–9](#), 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 specify the key property needed to do a lookup based on the cache key.

Consider two streams *S* and *C* with schemas (*id*, *group*, *value*) where the cache key is *id*. A valid query is:

```
select count(*) as n from S [now], C
where S.id = C.id
```

For instructions on specifying the cache key, see:

- [Section , "Specifying the Key Used to Index a 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).
- If the cache is a processor source, you connect the cache directly to the processor on the EPN as [Figure 13–1](#) shows.
- If the cache is a processor sink, you connect the processor to the cache using a channel as [Figure 13–2](#) shows.

### 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 , "Configuring an Oracle Event Processing Local Caching System and Cache"](#)
- [Section , "Configuring an Oracle Coherence Caching System and Cache"](#)
- [Section , "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 , "Creating Event Types"](#).

2. Specify the key properties for the data in the cache.

For instructions on specifying the cache key, see:

- [Section , "Specifying the Key Used to Index a 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>

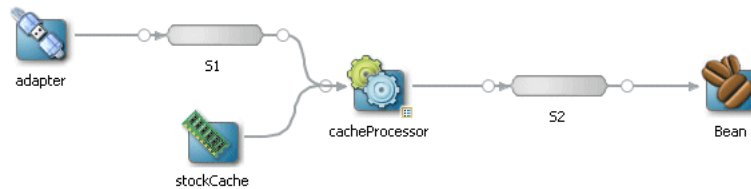
```

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:
  - a. If the cache is a processor source: you connect the cache directly to the processor on the EPN as [Figure 13-1](#) shows.

**Figure 13-1 Cache as Processor Source**



Update the wlevs:processor element a wlevs:cache-source child element that references the cache. For example:

```

<wlevs:channel id="S1"/>

<wlevs:processor id="cacheProcessor">
    <wlevs:source ref="S1">
        <wlevs:cache-source ref="cache-id">
    </wlevs:processor>

```

In the example, the processor will have data pushed to it from the S1 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.

- b. If the cache is a processor sink: you must connect the processor to the cache using a channel on the EPN (that is, there must be a channel between the processor and the cache sink) as [Figure 13-2](#) shows.

**Figure 13-2 Cache as Processor Sink**



In this case, the application assembly file looks like this:

```

<wlevs:channel id="channel1" event-type="StockTick">
    <wlevs:listener ref="processor" />

```

```
</wlevs:channel>
<wlevs:processor id="processor">
  <wlevs:listener ref="channel2" />
</wlevs:processor>
<wlevs:channel id="channel2" event-type="StockTick">
  <wlevs:listener ref="cache-id" />
</wlevs:channel>
```

## Accessing a Cache From an EPL Statement

You can reference a cache from an Event Processing Language (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.

---

---

**Note:** The EPL language is deprecated. For new development, use Oracle CQL. For information on accessing a cache from CQL, see [Section , "Accessing a Cache from an Oracle CQL Statement"](#).

---

---

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 Event Processing *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 , "Specifying the Key Used to Index a 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 Event Processing 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.

### 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 , "Configuring an Oracle Event Processing Local Caching System and Cache"](#)
- [Section , "Configuring an Oracle Coherence Caching System and Cache"](#)
- [Section , "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 , "Creating Event Types"](#).
2. Specify the key properties for the data in the cache. There are a variety of ways to do this; see
  - [Section , "Specifying the Key Used to Index a 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 be a valid type name in the event type repository.

This attribute is required only if the cache is referenced in an EPL query. This is because the query processor needs to know the type of events in the cache.

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.

## 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 Event Processing 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) {...}
}

```

## 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 [Section , "HelloWorld 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 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 Event Processing 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) {...}
}

```

## 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 bean with the `ref` attribute:

```
<wlevs:processor id= "tradeProcessor">
  <wlevs:function ref="orderFunction"/>
</wlevs:processor>
```

Alternatively, you can specify the bean class in the `wlevs:function` element:

```
<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>
```

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 Event Processing 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) {...}
}

```

For more information on user-defined functions, see "Functions: User-Defined" in the *Oracle Fusion Middleware CQL Language Reference for Oracle Event Processing*.

## 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 Event Processing 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) {...}
}

```

For more information on user-defined functions, see "User-Defined Functions" in the *Oracle Fusion Middleware EPL Language Reference for Oracle Event Processing*.

## Accessing a Cache Using JMX

At runtime, you can access a cache programatically using JMX and the MBeans that Oracle Event Processing deploys for the caching systems and caches you define.

This section describes:

- [Section , "How to Access a Cache With JMX Using Oracle Event Processing Visualizer"](#)
- [Section , "How to Access a Cache With JMX Using Java"](#)

For more information, "Configuring JMX for Oracle Event Processing" in the *Oracle Fusion Middleware Administrator's Guide for Oracle Event Processing*

### How to Access a Cache With JMX Using Oracle Event Processing Visualizer

The simplest and least error-prone way to access a caching system or cache with JMX is to use the Oracle Event Processing Visualizer.

For more information, see "Server and Domain Tasks" in the *Oracle Fusion Middleware Visualizer User's Guide for Oracle Event Processing*.

### 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 Event Processing Visualizer (see [Section , "How to Access a Cache With JMX Using Oracle Event Processing Visualizer"](#)). Alternatively, you can access a caching system or cache with JMX using Java code that you write.

Oracle Event Processing 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 Event Processing server provides.

For more information, see "Configuring JMX for Oracle Event Processing" in the *Oracle Fusion Middleware Administrator's Guide for Oracle Event Processing*

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.



---

---

## Integrating Web Services

This chapter describes how to configure web services for use with Oracle Event Processing, including how to invoke services from an Oracle Event Processing application and expose an Oracle Event Processing application as a web service.

This chapter includes the following sections:

- [Understanding Oracle Event Processing and Web Services](#)
- [How to Invoke a Web Service From an Oracle Event Processing Application](#)
- [How to Expose an Oracle Event Processing Application as a Web Service](#)

### Understanding Oracle Event Processing and Web Services

You can integrate an Oracle Event Processing application with other systems using Web Services.

Oracle Event Processing supports version 2.0 of the JAX-WS API standard using the Glassfish reference implementation of JAX-WS 2.0, including:

- JAX-WS 2.0 (Java API for XML Web Services, defined in JSR 224)
- WS-I Basic Profile 1.1
- WS-I Attachments Profile 1.0 (SOAP Messages with Attachments)
- WS-I Simple SOAP Binding Profile 1.0
- SOAP 1.1 and 1.2 (Simple Object Access Protocol)
- MTOM (Message Transmission Optimization Mechanism)
- WSDL 1.1 (Web Services Definition Language)
- JAXB 2.0 (Java API for XML Binding, references through a separate JAXB module)
- SAAJ 1.3 (SOAP with Attachments API for Java)

### How to Invoke a Web Service From an Oracle Event Processing Application

This procedure describes how to create an Oracle Event Processing application that invokes a Web Service. In this scenario, the Oracle Event Processing application is the Web Service client.

**To invoke a Web Service from an Oracle Event Processing application:**

1. Create or obtain the WSDL for the Web Service.

In this example, assume the use of a WSDL named `EchoService.WSDL`.

2. Generate the compiled `.class` files you will use to invoke the Web Service (in practice, the command should be on one line):

```
java -cp OCEP_HOME_DIR/modules/com.bea.core.ws.glassfish.jaxws.tools_9.0.0.0.jar com.sun.tools.ws.WsImport EchoService.WSDL
```

Where `ORACLE_CEP_HOME` refers to the directory in which you installed Oracle Event Processing (such as `/oracle_home`).

3. Archive the generated `.class` files within the Oracle Event Processing application JAR file.

For more information, see [Section , "Managing Libraries and Other Non-Class Files in Oracle Event Processing Projects"](#).

4. Export the Web-Services Java packages for the client-code in the `MANIFEST.MF` file using the `Export-Package` header:

```
Export-Package: com.oracle.ocep.sample.echoService;
```

For more information, see [Section , "How to Export a Package"](#).

5. Import the following packages to the Oracle Event Processing application in the `MANIFEST.MF` file using the `Import-Package` header:

```
Import-Package: com.ctc.wstx.stax, com.sun.xml.bind.v2, com.sun.xml.messaging.saaj.soap, com.sun.xml.messaging.saaj.soap.ver1_1, com.sun.xml.ws, javax.jws, javax.xml.bind, javax.xml.bind.annotation, javax.xml.namespace, javax.xml.soap, javax.xml.transform, javax.xml.transform.stream, javax.xml.ws, javax.xml.ws.spi, org.xml.sax, weblogic.xml.stax;
```

For more information, see [Section , "Managing Libraries and Other Non-Class Files in Oracle Event Processing Projects"](#).

6. Use the client-code to invoke the Web Service as in any other Java application:

```
EchoService service = new EchoService(); EchoPort port = service.getEchoServicePort(); String echo = port.echo("foo");
```

## How to Expose an Oracle Event Processing Application as a Web Service

This procedure describes how to expose an Oracle Event Processing application as a Web Service. In this scenario, the Oracle Event Processing application is the Web Service provider.

### To expose an Oracle Event Processing application as a Web service:

1. Create or obtain the WSDL for the Web Service.

In this example, assume the use of a WSDL named `EchoService.WSDL`.

**2. Implement the service.**

Consider using `java.jws` annotations `@WebService` and `@WebMethod`.

**3. Add a `bea-jaxws.xml` file to your application bundle as [Example 14-1](#) shows. [Table 14-1](#) describes the attributes in this file.**

**Example 14-1 `bea-jaxws.xml` File**

```
<endpoints>
  <endpoint>
    <name>EchoService</name>
    <implementation-class>
      com.bea.wlevs.test.echo.impl.EchoServiceImpl
    </implementation-class>
    <url-pattern>/echo</url-pattern>
    <wsdl-location>
      /META-INF/wsdl/echo.wsdl
    </wsdl-location>
    <service-name>
      {http://wsdl.oracle.com/examples/cep/echo}EchoService
    </service-name>
    <port-name>
      {http://wsdl.oracle.com/examples/cep/echo}EchoServicePort
    </port-name>
  </endpoint>
</endpoints>
```

**Table 14-1 `bea-jaxws.xml` File Attributes**

Attribute	Description
<code>name</code>	The name of the web service.
<code>implementation-class</code>	The class that implements the service.
<code>url-pattern</code>	The url pattern to access the web service.
<code>wsdl-location</code>	Relative path to the wsdl in the bundle.
<code>service-name</code>	QName of the service.
<code>port-name</code>	QName of the port.

For more information, see [Section , "Managing Libraries and Other Non-Class Files in Oracle Event Processing Projects"](#).

**4. Reference the `bea-jaxws.xml` file in the `MANIFEST.MF` file using the `BEA-JAXWS-Descriptor` header:**

```
BEA-JAXWS-Descriptor: META-INF/bea-jaxws.xml;
```

For more information, see [Section , "Managing Libraries and Other Non-Class Files in Oracle Event Processing Projects"](#).

**5. Import the following packages to the Oracle Event Processing application in the `MANIFEST.MF` file using the `Import-Package` header:**

```
Import-Package: com.ctc.wstx.stax,
  com.sun.xml.bind.v2,
  com.sun.xml.messaging.saaj.soap,
  com.sun.xml.ws,
```

```
javax.jws,  
javax.xml.bind,  
javax.xml.bind.annotation,  
javax.xml.namespace,  
javax.xml.soap,  
javax.xml.transform,  
javax.xml.transform.stream,  
javax.xml.ws,  
javax.xml.ws.spi,  
org.xml.sax,  
weblogic.xml.stax;
```

For more information, see [Section , "Managing Libraries and Other Non-Class Files in Oracle Event Processing Projects"](#).

6. Add a `glassfish-ws` element to the Oracle Event Processing server `DOMAIN_DIR/config/config.xml` file that describes your Oracle Event Processing domain, where `DOMAIN_DIR` refers to your domain directory:

```
<glassfish-ws>  
  <name>JAXWS</name>  
  <http-service-name>JettyServer</http-service-name>  
</glassfish-ws>
```

---

## Integrating an External Component Using a Custom Adapter

This chapter describes how to implement and configure your own Oracle Event Processing adapters for sending and receiving event data between your application and external components, including how to pass login credentials from an adapter.

You can develop adapters to exchange event data with external components that aren't supported by the adapters included with Oracle Event Processing. For more information about included adapters, see [Chapter 11, "Integrating the Java Message Service"](#), [Chapter 12, "Integrating an HTTP Publish-Subscribe Server"](#), and [Chapter 21, "Testing Applications With the Load Generator and csvgen Adapter"](#).

This chapter includes the following sections:

- [Overview of Custom Adapters](#)
- [Implementing a Custom Adapter](#)
- [Implementing Support for Thread and Work Management](#)
- [Passing Login Credentials from an Adapter to a Data Feed Provider](#)
- [Configuring a Custom Adapter](#)
- [Creating a Custom Adapter Factory](#)

### Overview of Custom Adapters

You can develop custom adapters to exchange event data with external components that aren't supported by adapters included with Oracle Event Processing.

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.
- Use messaging systems, such as TIBCO Rendezvous.
- Use a socket connection to the customer's own data protocol.

Adapters you build are likely to go at the beginning of an EPN, where they receive data, or at the end of an EPN, where they send event data elsewhere.

With a custom adapter, you can:

- Receive raw event data from an external component, then convert the data into event type instances that your application can use to process the events. If

necessary, your implementation can authenticate itself with the external component.

One of the main roles of an adapter is to convert incoming data, such as a market data feed, into Oracle Event Processing events. These events are then passed to other components in the EPN.

- Receive event type instances from within the event processing network, then convert the data to a form that's consumable by an external component. As the exit point of an application, an adapter receives events from another stage in the EPN, converts the event's data into something that an external application can read, and then sends it out.

You implement an adapter by creating an adapter class in Java. An adapter class implements Oracle Event Processing interfaces through which it can create, send, or receive events. For more information, see [Section , "Implementing a Custom Adapter"](#).

You add the adapter to the EPN by configuring it in the EPN assembly file. You can further specify runtime-editable configuration settings by configuring the adapter in a component configuration file. For more information, see [Section , "Configuring a Custom Adapter"](#).

The samples in the following list include source and sink example code

- The Foreign Exchange (FX) sample includes three adapters that read 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. For more information, see [Section , "Foreign Exchange \(FX\) Example"](#).
- The Oracle Spatial sample includes an adapter that reads data from a file and creates event type instances from the data. For more information, see [Section , "Oracle Spatial Example"](#).

## Implementing a Custom Adapter

You implement a custom adapter by writing Java code that can communicate with the external component you're integrating. An adapter class implementation also includes code to receive or send event type instances, depending on where in an EPN the adapter is intended to go.

The implementation will likely use several classes from the Oracle Event Processing API. For Javadoc reference on those, see the *Oracle Fusion Middleware Java API Reference for Oracle Event Processing*. For example code, see [Example 15-1, "High-Level View of Input Adapter Class"](#).

The following lists the high-level steps you might typically take when creating a custom adapter:

1. Implement a Java class that can communicate with the external component for which the adapter is intended. Of course, the specifics of your code will depend heavily on how the external component sends or receives the data your adapter is handling.
2. If the adapter will support being suspended and resumed (for example, when it is undeployed and deployed), you can implement interfaces to handle these events. For more information, see [Section , "Suspending and Resuming Adapter Event Processing"](#).
3. You might want be able to improve the application's scalability with finer control over adapter threading. For more information, see [Section , "Improving Scalability with Multi-Threaded Adapters"](#).

4. In your Java code, implement the interfaces needed to support sending or receiving event type instances.
  - If your adapter will be sending events, such as events it has created from incoming raw event data (as an input adapter would), you will want to implement it as an event source. For more information, see [Section , "Implementing an Event Source."](#)
  - If your adapter will be receiving events from some other stage in the EPN, implement it as an event sink. For more information, see [Section , "Implementing an Event Sink."](#)
5. If your custom adapter must authenticate itself with a data feed provider, as it might if it will be an input adapter, write Java logic to pass login credentials to the component providing event data. For more information, see [Section , "Passing Login Credentials from an Adapter to a Data Feed Provider."](#)
6. Optionally create a factory class. You need to do this only if multiple applications are going to use instances of the custom adapter. For more information, see [Section , "Creating a Custom Adapter Factory."](#)

If you want to bundle the custom adapter in its own JAR file so that it can be shared among multiple applications, see [Section , "How to Assemble a Custom Adapter in its Own Bundle."](#)
7. Add the adapter to an event processing network by configuring it in an EPN assembly file. For more information, see [Section , "Configuring a Custom Adapter."](#)

## Example: Input Adapter Implementation

This section provides a high-level illustration of an input adapter that retrieves raw event data from a file, converts the data into events, then sends the events to a downstream stage in the EPN.

The code here is excerpted from the Oracle Spatial sample application. Be sure to see its source code for the full implementation. For more information, see [Section , "Oracle Spatial Example"](#).

[Example 15-1, "High-Level View of Input Adapter Class"](#) is not a real-world scenario, but it does illustrate a basic flow for an input adapter. The following provides an overview of the work done by this class:

- Through dependency injection, this class is injected with instances of classes with which it will do some of its work, including:
  - An `EventTypeRepository` instance (injected with the `setEventTypeRepository` method) with which to retrieve an instance of the event type specified in the adapter's configuration.
  - A `StreamSender` instance (injected with the `setEventSender` method) with which to send events it generates.
- Through the `setPath` and `setEventType` methods, the class is injected with property values specified in the adapter's EPN assembly file configuration.
- In implementing the `RunnableBean` interface, this class provides a `run()` method implementation that does the adapter's real work: retrieving raw event data, parsing the data into event type instances, and sending the new events to a downstream EPN stage.

**Example 15–1 High-Level View of Input Adapter Class**

```

package com.oracle.cep.sample.spatial;

import java.io.BufferedReader;
import java.io.File;
import java.io.FileReader;

import org.apache.commons.logging.Log;
import org.apache.commons.logging.LogFactory;

import com.bea.wlevs.ede.api.EventProperty;
import com.bea.wlevs.ede.api.EventRejectedException;
import com.bea.wlevs.ede.api.EventType;
import com.bea.wlevs.ede.api.EventTypeRepository;
import com.bea.wlevs.ede.api.RunnableBean;
import com.bea.wlevs.ede.api.StreamSender;
import com.bea.wlevs.ede.api.StreamSink;
import com.bea.wlevs.ede.api.StreamSource;
import com.bea.wlevs.util.Service;
import java.lang.RuntimeException;

public class BusStopAdapter implements RunnableBean, StreamSource, StreamSink
{
    static final Log s_logger =
        LogFactory.getLog("BusStopAdapter");

    private String m_filePath;
    private String m_eventTypeName;
    private EventType m_eventType;
    private StreamSender m_eventSender;
    private boolean m_stopped;

    private int m_repeat = 1;
    private EventTypeRepository m_etr = null;

    public BusStopAdapter()
    {
        super();
    }

    /**
     * Called by the server to pass in the path
     * to the file with bus stop data.
     *
     * @param path The value specified for the path
     * property in the adapter's configuration
     * in the EPN assembly file.
     */
    public void setPath(String path) throws RuntimeException
    {
        // Code to create a File instance from the path. This
        // File object will be used to retrieve event data
        // from the file.
    }

    /**
     * Called by the server to pass in the name of the event
     * type to which event data should be bound.
     *
     * @param path The value specified for the path

```



```

    * property in the adapter's configuration
    * in the EPN assembly file.
    */
public void setEventType(String typ)
{
    m_eventTypeName = typ;
}

/**
 * Called by the server to set an event type
 * repository instance that knows about event
 * types configured for this application.
 *
 * This repository instance will be used to retrieve an
 * event type instance that will be populated
 * with event data retrieved from the event data file.
 *
 * @param etr The event repository.
 */
@Service(filter = EventTypeRepository.SERVICE_FILTER)
public void setEventTypeRepository(EventTypeRepository etr)
{
    m_etr = etr;
}

/**
 * Executes to retrieve raw event data and
 * create event type instances from it, then
 * sends the events to the next stage in the
 * EPN.
 *
 * This method, implemented from the RunnableBean
 * interface, executes when this adapter instance
 * is active.
 */
public void run()
{
    if (m_etr == null)
    {
        throw new RuntimeException("EventTypeRepository is not set");
    }

    // Get the event type from the repository by using
    // the event type name specified as a property of
    // this adapter in the EPN assembly file.
    m_eventType = m_etr.getEventType(m_eventTypeName);
    if (m_eventType == null)
    {
        throw new RuntimeException("EventType(" +
            m_eventType + ") is not found.");
    }

    BufferedReader reader = null;

    System.out.println("Sending " + m_eventType +
        " from " + m_filePath);
    while ((m_repeat != 0) && (!m_stopped))
    {
        try
        {

```

```

        reader = new BufferedReader(new FileReader(m_filePath));
    } catch (Exception e)
    {
        m_stopped = true;
        break;
    }
    while (!isStopped())
    {
        try
        {
            // Create an object and assign to it
            // an event type instance generated
            // from event data retrieved by the
            // reader.
            Object ev = null;
            ev = readLine(reader);
            if (ev == null)
            {
                reader.close();
                break;
            }

            // Send the newly created event type instance
            // to a downstream stage that is
            // listening to this adapter.
            m_eventSender.sendInsertEvent(ev);

        } catch (Exception e)
        {
            m_stopped = true;
            break;
        }
    }
}

/**
 * Called by the server to pass in a
 * sender instance that will be used to
 * send generated events to a downstream
 * stage.
 *
 * @param sender A sender instance.
 */
public void setEventSender(StreamSender sender)
{
    m_eventSender = sender;
}

/**
 * Returns true if this adapter instance has
 * been suspended, such as because an exception
 * occurred.
 */
private synchronized boolean isStopped()
{
    return m_stopped;
}

```

```

/**
 * Reads data from reader, creating event type
 * instances from that data. This method is
 * called from the run() method.
 *
 * @param reader Raw event data from a file.
 * @return An instance of the event type specified
 * as a property of this adapter.
 */
protected Object readLine(BufferedReader reader) throws Exception
{
    // Code to read raw event data and return an event type
    // instance from it.
}

/**
 * Called by the server to pass in an
 * insert event received from an
 * upstream stage in the EPN.
 */
@Override
public void onInsertEvent(Object event) throws EventRejectedException
{
    // Code to begin executing the logic needed to
    // convert incoming event data to event type instances.
}
}

```

## Implementing Support for Thread and Work Management

You can improve how your adapter's work is managed by implementing or configuring specific threading and work characteristics.

To improve scalability, for example, you might want to configure the adapter to be multi-threaded. To execute logic that responds well when the EPN is suspended or resumed, you can implement interfaces available in the Oracle Event Processing API.

For more information, see the following sections:

- [Section , "Improving Scalability with Multi-Threaded Adapters"](#)
- [Section , "Suspending and Resuming Adapter Event Processing"](#)

### Improving Scalability with Multi-Threaded Adapters

You can implement or configure an adapter to use one or more threads for reading from its data source. For example, a multi-threaded adapter might improve performance if its event-processing work is expensive.

Note that when an adapter is single-threaded, event order is guaranteed. Event order is not guaranteed in a multi-threaded adapter.

The simplest way to manage threading is to configure the adapter with a work manager. A work manager is a server feature through which your application can prioritize the execution of its work. You can specify a dedicated work manager used only by the adapter or you can share a work manager among several components such as other adapters.

For more information, see:

- [Section , "work-manager"](#)

- [Section , "EventPartitioner"](#)

You can also create a single-threaded adapter by implementing the `com.bea.wlevs.ede.api RunnableBean` interface. In your implementation of its `run()` method, you put the code that reads incoming data, converts it into Oracle Event Processing event type instances, and then send the events to the next stage in the EPN. For reference information on this interface, see the *Oracle Fusion Middleware Java API Reference for Oracle Event Processing*.

## Suspending and Resuming Adapter Event Processing

You can implement your adapter's Java class so that the adapter supports having its work be suspended or resumed by the server. For example, when an event processing network that the adapter part of is suspended, you might want the adapter to stop processing events. When the EPN's work is resumed, other code in the adapter can resume processing events. Supporting these cases might also mean managing resources that the adapter acquires in order to do its work.

To support being suspended or resumed, an adapter implements interfaces in the Oracle Event Processing API. These include the interfaces described in [Table 15–1, "Interfaces to Support Suspending and Resuming an Adapter"](#):

**Table 15–1 Interfaces to Support Suspending and Resuming an Adapter**

Interface	Description
<code>com.bea.wlevs.ede.api.SuspendableBean</code>	Implement this to provide logic that executes when the EPN is suspended. The interface's <code>suspend</code> method, you might suspend resources or stop processing events.
<code>com.bea.wlevs.ede.api.ResumeableBean</code>	Implement this to provide logic that executes when the EPN resumes work. In your implementation of the <code>beforeResume</code> method, you can provide code that should execute before the adapter's work resumes.

For reference information on these interfaces, see the *Oracle Fusion Middleware Java API Reference for Oracle Event Processing*.

## Passing Login Credentials from an Adapter to a Data Feed Provider

If your adapter accesses an external data feed, the adapter might need to pass login credentials, such as a 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 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 first 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 an 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 , "How to Pass Static Login Credentials to the Data Feed Provider"](#)
- [Section , "How to Pass Dynamic Login Credentials to the Data Feed Provider"](#)
- [Section , "How to Access Login Credentials From an Adapter at Runtime"](#)

For more information, see [Chapter 26, "Extending Component Configuration"](#).

## How to Pass Static Login Credentials to the Data Feed Provider

This section describes how to pass login credentials that you configure statically in the EPN assembly file.

### To pass static credentials to the data feed provider

1. Open a command window and set your environment as described in [Section , "Setting Your Development Environment."](#)
2. Change to the directory that contains the EPN assembly file for your application.
3. Edit the EPN assembly XML 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. In the command prompt, use the `encryptMSAConfig` command to encrypt the value of the `password` element in the EPN assembly file:

```
prompt> ORACLE_CEP_HOME/occep_11.1/bin/encryptMSAConfig . epn_assembly_file
aesinternal.dat_file
```

This command includes the following parts:

- `ORACLE_CEP_HOME` refers to the main directory into which you installed Oracle Event Processing, 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 `aesinternal.dat_file` parameter refers to the location of the `.aesinternal.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 Fusion Middleware Administrator's Guide for Oracle Event Processing*.

After you run the command, the value of the `password` element of the EPN assembly file will be encrypted.

6. Update your adapter Java code to access the login credentials properties you have just configured and decrypt the password.

See [Section , "How to Access Login Credentials From an Adapter at Runtime."](#)

7. Edit the `MANIFEST.MF` file of the application and add the `com.bea.core.encrypted` package to the `Import-Package` header. See [Section , "Creating the MANIFEST.MF File."](#)
8. Re-assemble and deploy your application as usual. See [Chapter 23, "Assembling and Deploying Oracle Event Processing Applications."](#)

## How to Pass Dynamic Login Credentials to the Data Feed Provider

This section describes how to pass login credentials that you configure dynamically by extending the configuration of the adapter.

### To pass dynamic login credentials 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 [Chapter 26, "Extending Component Configuration"](#) for detailed instructions.

2. Open a command window and set your environment as described in [Section , "Setting Your Development Environment."](#)
3. Change to the directory that contains the component configuration XML file for your adapter.
4. Update this component configuration XML file by adding the required login credentials using the `<user>` and `<password>` elements you defined in the configuration extension. 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 of the `password` element in the adapter configuration file:

```
prompt> ORACLE_CEP_HOME/ocep_11.1/bin/encryptMSAConfig . adapter_config_file
aesinternal.dat_file
```

This command includes the following parts:

- `ORACLE_CEP_HOME` refers to the main directory into which you installed Oracle Event Processing, 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 `aesinternal.dat_file` parameter refers to the location of the `.aesinternal.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 Fusion Middleware Administrator's Guide for Oracle Event Processing*.

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 , "How to Access Login Credentials From an Adapter at Runtime."](#)
8. Edit the `MANIFEST.MF` file of the application and add the `com.bea.core.encrypted` package to the `Import-Package` header. See [Section , "Creating the MANIFEST.MF File."](#)
9. Re-assemble and deploy your application as usual. See [Chapter 23, "Assembling and Deploying Oracle Event Processing Applications."](#)

## How to Access Login Credentials From an Adapter at Runtime

This section describes how update your custom adapter Java code to dynamically get the user and password values from the extended adapter configuration, and then use the `com.bea.core.encrypted.EncryptionService` API to decrypt the encrypted password.

### To access login credential properties from an adapter at runtime:

1. Import the additional APIs that you will need to decrypt the encrypted password:

```
import com.bea.core.encrypted.EncryptionService;
import com.bea.core.encrypted.EncryptionServiceException;
import com.bea.wlevs.util.Service;
```

2. Use the `@Service` annotation to get a reference to the `EncryptionService`:

```
private EncryptionService encryptionService;

@Service
public void setEncryptionService(EncryptionService encryptionService) {
```

```

        this.encryptionService = encryptionService;
    }

```

3. 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 , "Programming Access to the Configuration of a Custom Adapter or Event Bean"](#) for information about accessing the extended adapter configuration.

4. 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

```

5. Pass these credentials to the data feed provider using the vendor API.

## Configuring a Custom Adapter

When you create a custom adapter, you can add it to an EPN by configuring it in the EPN assembly file. You can also add adapter configuration to a component configuration file to support runtime configuration changes to certain features.

For more information, see the following sections:



- [Section , "Configuring a Custom Adapter in an EPN Assembly File"](#)
- [Section , "Configuring a Custom Adapter in a Component Configuration File"](#)

For a complete description of the configuration file, including registration of other components of your application, see [Section , "Creating EPN Assembly Files."](#)

## Configuring a Custom Adapter in an EPN Assembly File

In the EPN assembly file, you use the `wlevs:adapter` element to declare an adapter as a component in the event processor network. Note that the configuration code will differ for event beans created from a factory. For more information, see [Section , "Creating a Custom Adapter Factory"](#).

---

**Note:** The simplest way to create a custom adapter is using the Oracle Event Processing IDE for Eclipse adapter wizard. For more information, see [Section , "How to Create an Adapter Node"](#).

---

You can also use `wlevs:instance-property` child elements of `wlevs:adapter` to set any static properties in the adapter. Static properties are those that you will not dynamically change after the adapter is deployed.

In the following example, the `BusStopAdapter` class is configured as an adapter to set properties implemented as methods in the class (`setPath`, `setEventType`, and `setBuffer`):

```
<wlevs:adapter id="BusStopAdapter"
  class="com.oracle.cep.sample.spatial.BusStopAdapter" >
  <wlevs:instance-property name="path" value="bus_stops.csv"/>
  <wlevs:instance-property name="eventType" value="BusStop"/>
  <wlevs:instance-property name="buffer" value="30.0"/>
</wlevs:adapter>
```

You reference an adapter that is an event sink by using the `wlevs:listener` element. In the following example, a `BusPositionGen` CSV adapter sends events to the `BusStopAdapter`:

```
<wlevs:adapter id="BusPositionGen" provider="csvgen">
  <!-- Code omitted -->
  <wlevs:listener ref="BusStopAdapter"/>
</wlevs:adapter>
```

## Configuring a Custom Adapter in a Component Configuration File

You can add configuration for an adapter in a component configuration file. Configuration you add here is available to be updated at runtime. Adding configuration in a component configuration file assumes that you have added the adapter to the EPN by configuring it in the EPN assembly file (see [Section , "Configuring a Custom Adapter in an EPN Assembly File"](#) for more information).

You can also create a separate component configuration XML file as needed according to your development environment. For example, if your application has more than one custom event bean, you can create separate XML files for each, or create a single XML file that contains the configuration for all custom event beans, or even all components of your application (beans, adapters, processors, and streams).

In the following example, a `BusStopAdapter` adapter is configured for event recording. Each adapter configuration should have a separate adapter child element of the config element.

```
<?xml version="1.0" encoding="UTF-8"?>
<wlevs:config
  xmlns:wlevs="http://www.bea.com/ns/wlevs/config/application">
  <adapter>
    <name>BusStopAdapter</name>
    <record-parameters>
      <dataset-name>spatial_sample</dataset-name>
      <event-type-list>
        <event-type>BusPos</event-type>
        <event-type>BusStop</event-type>
        <event-type>BusPosEvent</event-type>
        <event-type>BusStopArrivalEvent</event-type>
        <event-type>BusStopPubEvent</event-type>
        <event-type>BusStopPubEvent</event-type>
      </event-type-list>
      <batch-size>1</batch-size>
      <batch-time-out>10</batch-time-out>
    </record-parameters>
  </event-bean>
</wlevs: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 Event Processing knows to which particular custom adapter component in the EPN assembly file this configuration applies.

You can also extend component configuration with your own elements. For more information, see [Section 26, "Extending Component Configuration"](#).

For more information, see:

- [Section , "Component Configuration Schema wlevs\\_application\\_config.xsd"](#)

## Creating a Custom Adapter Factory

You can use a single adapter implementation in multiple event processing networks by implementing and configuring an adapter factory. The factory class provides adapter instances for the applications that request one.

For detail on the APIs described here, see the *Oracle Fusion Middleware Java API Reference for Oracle Event Processing*.

Creating an adapter factory involves the following steps:

1. In your adapter class, implement the `com.bea.wlevs.ede.api.Adapter` interface so that the adapter can be returned by the factory. This is a marker interface, so there are no methods to implement.

```
public class BusStopAdapter implements Adapter, StreamSource {
    // Adapter implementation code.
}
```

2. Implement an adapter factory class to create and return instances of the adapter.

Your adapter factory class must implement the `com.bea.wlevs.ede.api.AdapterFactory` interface. In your implementation of its `create` method, create an instance of your adapter.

```
import com.oracle.cep.sample.spatial.BusStopAdapter;
import com.bea.wlevs.ede.api.AdapterFactory;

public class BusStopAdapterFactory implements AdapterFactory {
    public BusStopAdapterFactory() {}
    public synchronized BusStopAdapter create()
        throws IllegalArgumentException {

        // Your code might have a particular way to create the instance.
        return new BusStopAdapter();

    }
}
```

**3.** In an EPN assembly file, configure the factory class.

You register factories in the EPN assembly file using the `wlevs:factory` element, as shown in the following example:

```
<wlevs:factory provider-name="busStopAdapterProvider"
    class="com.oracle.cep.sample.spatial.BusStopAdapterFactory"/>
```

If you need to specify service properties, then you must also 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 all of Oracle Event Processing. 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 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 your specific adapter class in the Spring application context.

```
<osgi:service interface="com.bea.wlevs.ede.api.AdapterFactory">
    <osgi:service-properties>
        <entry key="type" value="busStopAdapterProvider"/>
    </osgi:service-properties>
    <bean class="com.oracle.cep.sample.spatial.BusStopAdapterFactory" />
</osgi:service>
```

**4.** In applications that will use instances of the adapter, configure the adapter by specifying the configured factory as a provider (rather than specifying the adapter by its class name), as shown in the following example:

```
<wlevs:adapter id="BusStopAdapter"
    provider="busStopAdapterProvider">
    // ...
</wlevs:adapter>
```



---

---

## Handling Events with Java

This chapter describes how to implement the Oracle Event Processing interfaces needed for a Java class to act as an event sink and event source, receiving and sending events in an event processing network (EPN). It also describes how to configure a Java class as an Oracle Event Processing event bean or Spring bean.

Whether you are writing new logic in Java or wanting to incorporate existing Java code, Oracle Event Processing provides several ways to add Java code to your application. Your options include where and how the code executes, as well as how you configure it.

This chapter includes the following sections:

- [Roles for Java Code in an Event Processing Network](#)
- [Handling Events with Sources and Sinks](#)
- [Configuring Java Classes as Beans](#)

### Roles for Java Code in an Event Processing Network

Whether you are writing new functionality or have existing logic in Java that you want to incorporate into an Oracle Event Processing application, places where you code might go depending on its role.

Note that many Oracle Event Processing applications have no need for Java code at all. For example, an application's logic might be captured in Oracle CQL alone.

Use the following descriptions of roles for Java code to find the best place for your code.

- **Java classes as beans to handle events passing through an event processing network (EPN)**

You can write Java code to handle events as they flow through the EPN, receiving events, sending them, or both. For example, you could add a class that supports both receiving and sending events, using it as intermediate logic in the EPN. There, it could retrieve data from events it receives, then create a new kind event from the data for use by a particular downstream component.

For more information on implementing Java classes that receive and send events, see [Section , "Configuring Java Classes as Beans"](#).

- **Custom adapters to integrate external components for incoming or outgoing events**

You can implement a Java class as logic in a custom adapter designed to interact with an external component that is not supported by adapters included with

Oracle Event Processing. For example, you could implement an adapter that is able to receive events from an event source that isn't supported by the included JMS or HTTP Publish-Subscribe adapters.

The Java code in custom adapters implements the event source and event sink functionality discussed in this chapter -- see [Section , "Handling Events with Sources and Sinks"](#). For more information about other aspects of custom adapters, see [Chapter 15, "Integrating an External Component Using a Custom Adapter"](#).

- **Java functions to enhance functionality in Oracle CQL code**

You can call Java methods from Oracle CQL code to augment Oracle CQL with your own logic. For example, if you have written a Java class with methods that perform calculations on data such as that your application will receive as event data, you might call methods of that class within Oracle CQL code as part of a `select` statement.

For more information, see "User-Defined Functions" in the *Oracle Fusion Middleware CQL Language Reference for Oracle Event Processing*

- **JavaBean class as an event type that represents event data**

You can create an event type -- the vehicle for carrying event data through your application -- by implementing the type as a JavaBean class. Although there are alternatives, this is the best practice approach.

For more information, see [Section , "Creating an Oracle Event Processing Event Type as a JavaBean"](#).

## Handling Events with Sources and Sinks

When you write Java code designed to handle events, you create a class that is an event sink or event source. An event sink is able to receive events as they flow through an event processing network. An event source can create events and send them along to a downstream stage in the EPN. For example, you might create a class that receive events, does something with their data, then send them along to the next stage.

Event sinks and sources implement particular interfaces provided by the Oracle Event Processing API. An event sink implements interfaces that include methods through which the Oracle Event Processing server can pass into it event type instances. An event source implements an interface that includes a method through which it receives an object with which to send events. A single class can implement either or both kinds of functionality, depending on its role in the event processing network (EPN).

---

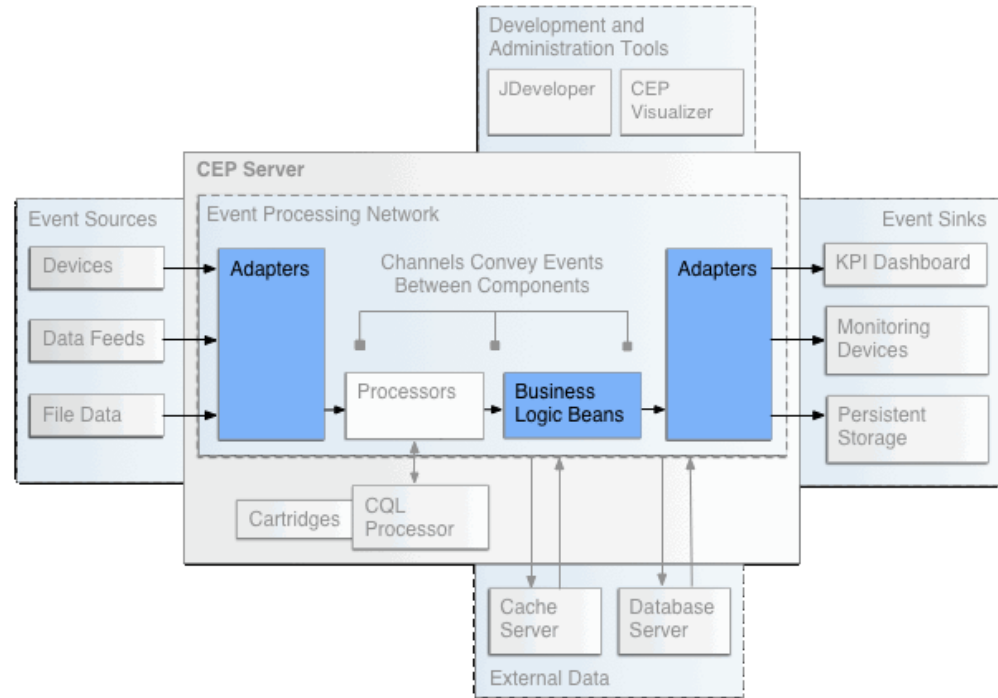
---

**Note:** This section assumes that you are familiar with streams and relations and the differing ways they represent events moving through an event processing network. If not, be sure to read [Section , "Overview of Events, Streams and Relations"](#).

---

---

Common places for event-handling Java code are adapters and beans. An adapter is a part of the EPN that communicates with external components. It is designed to receive external event data and create events from the data, or to receive internal events and send their data along to another component outside the EPN. Writing event handling code, as described in this chapter, is an important part of creating an adapter. For more on creating adapters, see [Chapter 15, "Integrating an External Component Using a Custom Adapter"](#).



Like an adapter, a bean can handle events. But you typically add a bean in the midst of an EPN so that events will pass into and out of it. For example, you might want to add a bean whose Java code executes application logic in response to data in events passing through it.

The following procedure describes the typical steps for creating a Java class that receives and sends events.

1. Implement the interfaces needed to receive or send events. Your options for developing Java logic as an EPN component are as follows:
  - To create a class that can receive events as they pass through the EPN, implement interfaces that make the class an event sink. Those interfaces include `StreamSink` or `RelationSink` for receiving single events, and `BatchStreamSink` or `BatchRelationSink` for receiving batches of events. For more information, see [Section , "Implementing an Event Sink"](#).
  - To create a class that can send events to other parts of the EPN, implement interfaces that make the class an event source. Those interfaces include `StreamSource` or `RelationSource` for sending single events, and `BatchStreamSource` or `BatchRelationSource` for sending batches of events. For more information, see [Section , "Implementing an Event Source"](#).
2. Configure the class so that you can add it to the EPN where it belongs.
  - If the class will be part of an adapter, you will want to finish the adapter implementation, then configure it as described in [Chapter 15, "Integrating an External Component Using a Custom Adapter"](#).
  - If the class will be added to the EPN as a bean, you will want to configure it as described in [Section , "Configuring Java Classes as Beans"](#).

---



---

**Note:** This procedure assumes that the custom event bean is bundled in the same application JAR file that contains the other components of the EPN, such as the processor, streams, and business logic POJO. If you want to bundle the custom event bean in its own JAR file so that it can be shared among multiple applications, see [Section , "How to Assemble an Event Bean in its Own Bundle."](#)

---



---

## Implementing an Event Sink

You can create a Java class that is able to receive events as they pass through an event processing network. A component that can receive events is an event sink. You might create an event sink, for example, to receive events in the midst of an event processing network, with logic for responding to each event's content.

A Java class that is an event sink implements one of the interfaces described in this section. Each of these interfaces provides methods that the Oracle Event Processing server uses to pass events to the class as the events exit the EPN stage connected upstream of the class, typically a channel.

---



---

**Note:** For a step-by-step development example that includes creating a simple event sink, be sure to see [Chapter 8, "Walkthrough: Assembling a Simple Application"](#).

---



---

The interfaces described here are intended to provide support for events arriving either as streams or relations. However, interfaces for relation support also support receiving events arriving as streams. As described in the following table, the interfaces are hierarchically related.

Interface	Description
<code>com.bea.wlevs.ede.api.StreamSink</code>	Implement this to receive events arriving sequentially as a stream.
<code>com.bea.wlevs.ede.api.RelationSink</code>	Implement this to receive events arriving sequentially as a relation. Extends <code>StreamSink</code> , so it also provides support for receiving events as a stream.
<code>com.bea.wlevs.ede.api.BatchStreamSink</code>	Implement this to support receiving batched events arriving as a stream. Events might arrive batched by timestamp if the channel they are coming from is configured to allow batching. Extends <code>StreamSink</code> , so it also provides support for receiving events unbatched.
<code>com.bea.wlevs.ede.api.BatchRelationSink</code>	Implement this to support receiving batched events arriving as a relation. Events might arrive batched by timestamp if the channel they are coming from is configured to allow batching. Extends <code>RelationSink</code> , so it also provides support for receiving events unbatched as either streams or relations.

### Implementing `StreamSink` or `BatchStreamSink`

A class that receives events as a stream will receive only events that are, from the Oracle Event Processing standpoint, "inserted." That's because in a stream, events are always appended to the end of a sequence. Events in a stream are also always received



in ascending time order, so that their timestamps have non-decreasing values from one event to the one that follows it. (The idea of non-decreasing timestamps allows for the possibility that the timestamp of one event can be *the same as* the timestamp of the event that precedes it, but not *earlier than* that preceding timestamp. It's either the same or later.)

As a result, the interfaces for support to receive events as a stream have one method each for receiving events. Contrast this with the interfaces for receiving events as a relation, which support receiving multiple kinds of events.

You implement the `StreamSink` interface if you expect your class to receive unbatched events as a stream. It has a single method, `onInsertEvent`, which the Oracle Event Processing server calls to pass in each event from the stream as it leaves the upstream stage that is connected to your class.

In [Example 16–1, "Implementing the StreamSink Interface"](#), a simple `StreamSink` implementation that receive stock trade events receives each event as an `Object` instance, then tests to see if the event is an instance of a particular event type. If it is, then the code retrieves values of properties known to be members of that type.

#### **Example 16–1 Implementing the StreamSink Interface**

```
public class TradeListener implements StreamSink {

    public void onInsertEvent(Object event) throws EventRejectedException {
        if (event instanceof TradeEvent){
            String symbolProp = ((TradeEvent) event).getSymbol();
            Integer volumeProp = ((TradeEvent) event).getVolume();
            // Code to do something with the property values.
        }
    }
}
```

You implement the `BatchStreamSink` interface if you expect your class to receive batched events as a stream. The interface has a single method, `onInsertEvents`, which the Oracle Event Processing server calls to pass in a collection of events received from the upstream stage. (The `BatchStreamSink` interface extends `StreamSink`, so can receive unbatched events also.)

For more information about event batching, see [Section , "Batch Processing Channels"](#).

#### **Implementing RelationSink or BatchRelationSink**

A class that receives events as a relation can receive any of the kinds of events possible in a relation: insert events, delete events, and update events. Unlike a stream, events in a relation are unordered and include events that have been updated or deleted by code that created or operated on the relation.

As a result, the interfaces for support to receive events as a relation have methods through which your class can receive insert, delete, or update events.

You implement the `RelationSink` interface if you expect your class to receive unbatched events as a relation. It has three methods (one inherited from the `StreamSink` interface, which it extends), `onInsertEvent`, `onDeleteEvent`, and `onUpdateEvent`. At runtime, the Oracle Event Processing server will call the appropriate method depending on which type of event is being received from the upstream channel connected to your class.

**Example 16–2 Implementing the RelationSink Interface**

```

public class TradeListener implements RelationSink {

    public void onInsertEvent(Object event) throws EventRejectedException {
        if (event instanceof TradeEvent){
            String symbolProp = ((TradeEvent) event).getSymbol();
            Integer volumeProp = ((TradeEvent) event).getVolume();
            // Do something with the inserted event.
        }
    }

    @Override
    public void onDeleteEvent(Object event) throws EventRejectedException {
        if (event instanceof TradeEvent){
            // Do something with the deleted event.
        }
    }

    @Override
    public void onUpdateEvent(Object event) throws EventRejectedException {
        if (event instanceof TradeEvent){
            // Do something with the updated event.
        }
    }
}

```

You implement the `BatchRelationSink` interface if you expect your class to receive batched events as a relation. It has an `onEvents` method designed to receive all three types of events from the batch in `java.util.Collection` instances:

```
onEvents(insertEvents, deleteEvents, updateEvents)
```

In addition, the interface extends the `RelationSink` interface to provide support for receiving unbatched events.

At runtime, the Oracle Event Processing server calls the appropriate method to pass in events received from the upstream stage connected to your class.

For more information about event batching, see [Section , "Batch Processing Channels"](#).

For complete API reference information about the Oracle Event Processing APIs described in this section, see the *Oracle Fusion Middleware Java API Reference for Oracle Event Processing*.

## Implementing an Event Source

You can create a Java class that is able to send events to a downstream stage in an event processing network. A component that can send events is an event source. You might create an event source, for example, to send events your Java code has created or altered from event data flowing through the EPN.

A Java class that is an event source implements one of the interfaces described in this section. Each of these interfaces provides a method used by the Oracle Event Processing server to pass into your class an instance of a sender class.

The sender instance your event source receives, in turn, implements one of the sender interfaces described in this section. The sender interfaces provide methods your code can call to send events as streams or relations, batched or unbatched, along to the downstream EPN stage that follows it, such as a channel.

The interfaces described here are intended to provide support for sending events either as streams or relations. However, interfaces for relation support also support sending events as streams.

**Table 16–1 Interfaces for Implementing an Event Source**

Interface	Description
<code>com.bea.wlevs.ede.api.StreamSource</code>	Implement this for the ability to send events as a stream. At runtime, the Oracle Event Processing server will inject an instance of a stream sender class.
<code>com.bea.wlevs.ede.api.RelationSource</code>	Implement this for the ability to send events as a relation or stream. At runtime, the Oracle Event Processing server will inject an instance of a relation sender class. Extends <code>StreamSource</code> , so it also provides support as a source of stream events.

The interfaces listed in [Table 16–2, "Interfaces Implemented by Sender Classes"](#) are implemented by sender classes your event source class receives from the Oracle Event Processing server.

**Table 16–2 Interfaces Implemented by Sender Classes**

Interface	Description
<code>com.bea.wlevs.ede.api.StreamSender</code>	Provides a method with which your code can send events as a stream.
<code>com.bea.wlevs.ede.api.RelationSender</code>	Provides methods with which your code can send events as a relation. Extends <code>StreamSender</code> , so it also provides support for sending events as a stream.
<code>com.bea.wlevs.ede.api.BatchStreamSender</code>	Provides a method with which your code can send batched events as a stream. You might send events batched by timestamp if the downstream stage to which you're sending them is a channel configured for batched events. Extends <code>StreamSender</code> , so it also provides support for sending events unbatched.
<code>com.bea.wlevs.ede.api.BatchRelationSender</code>	Provides a method with which your code can send batched events as a relation. You might send events batched by timestamp if the downstream stage to which you're sending them is a channel configured for batched events. Extends <code>RelationSender</code> , so it also provides support for sending events unbatched.

### Implementing StreamSource

A class that is a source of events as a stream should send only events that are, from the Oracle Event Processing standpoint, "inserted." Sending only inserted events models a stream, rather than a relation. Events sent from a stream source should also have non-decreasing timestamps from one event to the event that follows it. In other words, the timestamp of an event that follows another should either be the same as or later than the event that preceded it.

When you implement `StreamSource`, your code can send events batched or unbatched. Your implementation of the `StreamSource` `setEventSender` method will receive a sender instance that you can cast to one of the types described in [Table 16-2, "Interfaces Implemented by Sender Classes"](#). Your code should use the sender instance to send events as expected by the downstream stage to which the events will be going.

If your code is sending events to a channel that enables batching, you should use one of the batched event senders to batch events by timestamp before sending them. For more information, see [Section , "Batch Processing Channels"](#).

The sender instance also provides a `sendHeartbeat` method with which you can send a heartbeat if the receiving channel is configured to be application timestamped.

## Implementing RelationSource

A class that is a source of events as a relation can send insert, delete, and update events as expected by the downstream stage that is receiving the events.

When you implement `RelationSource`, your code can send events batched or unbatched. Your implementation of the `RelationSource` `setEventSender` method will receive a sender instance that you can cast to one of the types described in [Table 16-2, "Interfaces Implemented by Sender Classes"](#). Your code should use the sender instance to send events as expected by the downstream stage.

As you implement `RelationSource`, keep in mind the following constraints when using the sender instance your class receives:

- For `sendDeleteEvent`, you must send an instance of the same event type as that configured for the channel.
- For `sendInsertEvent`, a unique constraint violation exception will be raised and the input event discarded if an event with the same primary key is already in the relation.
- For `sendUpdateEvent`, an invalid update tuple exception will be raised and the input event will be discarded if an event with the given primary key is not in the relation.

In [Example 16-3, "Implementing the RelationSource Interface"](#), a simple `RelationSource` implementation receives a `StreamSender`, then casts the sender to a `RelationSender` in order to send events as a relation. This class creates a new `TradeEvent` instance from the event type configured in the repository, but the `sendEvents` method could as easily have received an instance as a parameter from another part of the code.

### **Example 16-3 Implementing the RelationSource Interface**

```
package com.oracle.cep.example.tradereport;

import com.bea.wlevs.ede.api.EventType;
import com.bea.wlevs.ede.api.EventTypeRepository;
import com.bea.wlevs.ede.api.RelationSender;
import com.bea.wlevs.ede.api.RelationSource;
import com.bea.wlevs.ede.api.StreamSender;
import com.bea.wlevs.util.Service;

public class TradeEventSource implements RelationSource {

    // Variables for event type repository and event sender. Both
    // will be set by the server.
    EventTypeRepository m_repos = null;
```

```

RelationSender m_sender = null;

// Called by the server to set the repository instance.
@Service
public void setEventTypeRepository(EventTypeRepository repos) {
    m_repos = repos;
}

// Called by the server to set the sender instance.
@Override
public void setEventSender(StreamSender sender) {
    // Cast the received StreamSender to a RelationSender
    m_sender = (RelationSender)sender;
}

/**
 * Sends events to the next EPN stage using the sender
 * received from the server. This code assumes that an event
 * instance isn't received from another part of the class,
 * instead creating a new instance from the repository.
 */
private void sendEvents(){
    EventType eventType = m_repos.getEventType("TradeEvent");
    TradeEvent tradeEvent = (TradeEvent)eventType.createEvent();
    m_sender.sendDeleteEvent(tradeEvent);
}
}

```

## Configuring Java Classes as Beans

When you write Java classes to handle events, you can add them to an EPN by configuring them in the EPN assembly file. You can configure a class as either a Spring bean or an Oracle Event Processing event bean.

---

**Note:** You can also add a class as part of an adapter, whose code receives or sends events when interacting with an external component. For more about configuring a class as an adapter, see [Chapter 15, "Integrating an External Component Using a Custom Adapter"](#).

---

Whether you configure your class as an event bean or Spring bean depends on your deployment context and the features you want to support. Essentially, however, you might want to configure the class as Spring bean if you're interested in taking advantage of an Spring features, such as when you're already using Spring elsewhere.

[Table 16–3, "Comparison of Event Beans and Spring Beans"](#) lists the features provided by event beans and Spring beans.

**Table 16–3 Comparison of Event Beans and Spring Beans**

Bean Type	Description
Event bean	<p>Useful as an EPN stage to actively use the capabilities of the Oracle Event Processing server container. An event bean:</p> <ul style="list-style-type: none"> <li>■ Is a type of Oracle Event Processing EPN stage.</li> <li>■ Can be monitored by the Oracle Event Processing monitoring framework.</li> <li>■ Can make use of the configuration metadata annotations.</li> <li>■ Can be set to record and play-back events that pass through it.</li> <li>■ Can participate in the Oracle Event Processing server bean lifecycle by specifying methods in its XML declaration, rather than by implementing Oracle Event Processing server API interfaces.</li> </ul>
Spring bean	<p>Useful for legacy integration to Spring. A Spring bean:</p> <ul style="list-style-type: none"> <li>■ Is useful if you have a Spring bean you want to add to an EPN.</li> <li>■ Is not a type of Oracle Event Processing EPN stage.</li> <li>■ Cannot be monitored by the Oracle Event Processing monitoring framework.</li> <li>■ Cannot use the configuration metadata annotations.</li> <li>■ Cannot be set to record and play back events that pass through it.</li> </ul>

For more details on each of the different bean types, see the following:

- [Section , "Configuring a Java Class as an Event Bean"](#)
- [Section , "Configuring a Java Class as a Spring Bean"](#)

## Configuring a Java Class as an Event Bean

You can configure a Java class as an Oracle Event Processing event bean in the EPN assembly file and the component configuration file. Configuring it as a event bean in the EPN assembly file will add the bean to the EPN with default settings. You can then configure it in the component configuration file to support runtime configuration changes to certain features.

For more information, see the following sections:

- [Section , "Configuring an Event Bean in an EPN Assembly File"](#)
- [Section , "Configuring an Event Bean in a Component Configuration File"](#)

For a complete description of the configuration file, including registration of other components of your application, see [Section , "Creating EPN Assembly Files."](#)

### Configuring an Event Bean in an EPN Assembly File

In an EPN assembly file, you use the `wlevs:event-bean` element to declare a custom event bean as a component in the event processor network. Note that the configuration code will differ for event beans created from a factory. For more information, see [Section , "Creating an Event Bean Factory"](#).

In the following example, the `TradeListener` class is configured as an event bean whose downstream connection is a channel with ID `BeanOutputChannel`:

```
<wlevs:event-bean id="TradeListenerBean"
  class="com.oracle.cep.example.tradereport.TradeListener">
  <wlevs:listener ref="BeanOutputChannel" />
</wlevs:event-bean>
```

You can also use a `wlevs:instance-property` child element to set any static properties in the bean. Static properties are those that you will not change after the bean is deployed.

For example, if your bean class has a `setThreshold` method, you can pass it the port number as shown:

```
<wlevs:event-bean id="TradeListenerBean"
  class="com.oracle.cep.example.tradereport.TradeListener">
  <wlevs:instance-property name="threshold" value="6000" />
  <wlevs:listener ref="BeanOutputChannel" />
</wlevs:event-bean>
```

You reference an event bean that is an event sink by using the `wlevs:listener` element. In the following example, a `TradeListenerBean` event bean receives events from a channel `ProcessorOutputChannel`:

```
<wlevs:channel id="ProcessorOutputChannel" >
  <wlevs:listener ref="TradeListenerBean" />
</wlevs:channel>
```

### Configuring an Event Bean in a Component Configuration File

You can add configuration for an event bean in a component configuration file. Configuration you add here is available to be updated at runtime. Adding configuration in a component configuration file assumes that you have added the event bean to the EPN by configuring it in the EPN assembly file (see [Section , "Configuring an Event Bean in an EPN Assembly File"](#) for more information).

You can also create a separate component configuration XML file as needed according to your development environment. For example, if your application has more than one custom event bean, you can create separate XML files for each, or create a single XML file that contains the configuration for all custom event beans, or even all components of your application (beans, adapters, processors, and streams).

In the following example, a `TradeListenerBean` event bean is configured for event recording. Each event bean configuration should have a separate `event-bean` child element of the `config` element.

```
<?xml version="1.0" encoding="UTF-8"?>
<wlevs:config
  xmlns:wlevs="http://www.bea.com/ns/wlevs/config/application">
  <event-bean>
    <name>TradeListenerBean</name>
    <record-parameters>
      <dataset-name>tradereport_sample</dataset-name>
      <event-type-list>
        <event-type>TradeEvent</event-type>
      </event-type-list>
      <batch-size>1</batch-size>
      <batch-time-out>10</batch-time-out>
    </record-parameters>
  </event-bean>
</wlevs:config>
```

Uniquely identify each event bean with the `name` child element. This name must be the same as the value of the `id` attribute in the `wlevs:event-bean` element of the EPN assembly file that defines the event processing network of your application. This is how Oracle Event Processing knows to which particular event bean component in the EPN assembly file this configuration applies.

You can also extend component configuration with your own elements. For more information, see [Section 26, "Extending Component Configuration"](#).

For a reference on the component configuration XML schema, see [Section , "Component Configuration Schema wlevs\\_application\\_config.xsd"](#).

## Creating an Event Bean Factory

You can use a single event bean implementation in multiple event processing networks by implementing and configuring an event bean factory. The factory class provides event bean instances for the applications that request one.

For detail on the APIs described here, see the *Oracle Fusion Middleware Java API Reference for Oracle Event Processing*

Creating an event bean factory involves the following steps:

1. In your event bean class, implement the `com.bea.wlevs.ede.api.EventBean` interface so that the bean can be returned by the factory. This is a marker interface, so there are no methods to implement.

```
public class TradeListener implements EventBean, StreamSink {
    // Bean implementation code.
}
```

2. Implement an event bean factory class to create and return instances of the event bean.

Your event bean factory class must implement the `com.bea.wlevs.ede.api.EventBeanFactory` interface. In your implementation of its `create` method, create an instance of your event bean.

```
import com.oracle.cep.example.tradereport.TradeListener;
import com.bea.wlevs.ede.api.EventBeanFactory;

public class TradeListenerFactory implements EventBeanFactory {
    public TradeListenerFactory() {
    }
    public synchronized TradeListener create()
        throws IllegalArgumentException {

        // Your code might have a particular way to create the instance.
        return new TradeListener();
    }
}
```

3. In an EPN assembly file, configure the factory class.

You register factories in the EPN assembly file using the `wlevs:factory` element, as shown in the following example:

```
<wlevs:factory provider-name="tradeListenerProvider"
    class="com.oracle.cep.example.tradereport.TradeListenerFactory"/>
```

If you need to specify service properties, then you must also 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 all of Oracle Event Processing. This means that if more than one application deployed to a given server is going to use the same event bean factory, be sure to register the factory only *once* as an OSGI service.



Add an entry to register the service as an implementation of the `com.bea.wlevs.ede.api.EventBeanFactory` interface. Provide a property, with the key attribute equal to `type`, and the name by which this event bean provider will be referenced. Finally, add a nested standard Spring bean element to register your specific event bean class in the Spring application context

```
<osgi:service interface="com.bea.wlevs.ede.api.EventBeanFactory">
  <osgi:service-properties>
    <entry key="type" value="tradeListenerProvider" />
  </osgi:service-properties>
  <bean class="com.oracle.cep.example.tradereport.TradeListenerFactory" />
</osgi:service>
```

4. In applications that will use instances of the event bean, configure the event bean by specifying the configured event bean factory as a provider (rather than specifying the bean by its class name), as shown in the following example:

```
<wlevs:event-bean id="TradeListenerBean"
  provider="tradeListenerProvider">
  ...
</wlevs:event-bean>
```

For more on bundling an event bean in its own bundle for reuse, see [Section , "How to Assemble an Event Bean in its Own Bundle"](#).

## Configuring a Java Class as a Spring Bean

You can configure a Java class as a Spring bean in order to include the class in an event processing network. This is a good option if you have an existing Spring bean that you want to incorporate into the EPN. Or you might simply want to have your Java code make use of Spring features.

A Spring bean is a Java class managed by the Spring framework. You add a class as a Spring bean by configuring it in the EPN assembly file using the standard bean element.

Keep in mind that a Spring bean is not a type of Oracle Event Processing stage. In other words, it cannot be monitored by the Oracle Event Processing monitoring framework, cannot use the configuration metadata annotations, and cannot be set to record and play-back events that pass through it.

In the EPN assembly file, you use the bean element to declare a custom Spring bean as a component in the event processor network. For example:

```
<bean id="TradeListenerBean"
  class="com.oracle.cep.example.tradereport.TradeListener">
</bean>
```

### Supporting Spring Bean Characteristics

In a Spring bean you are planning to add to an EPN, you can implement the various lifecycle interfaces. These include `InitializingBean`, `DisposableBean`, and the active interfaces, such as `RunnableBean`. The Spring bean event source can also use configuration metadata annotations such as `@Prepare`, `@Rollback`, and `@Activate`.



---

---

## Querying an Event Stream with Oracle CQL

This chapter describes how to configure Oracle Continuous Query Language (Oracle CQL) processors for Oracle Event Processing event processing networks. It includes information on configuring the processor's data source and optimizing performance.

This chapter includes the following sections:

- [Overview of Oracle CQL Processor Configuration](#)
- [Configuring an Oracle CQL Processor](#)
- [Configuring an Oracle CQL Processor Table Source](#)
- [Configuring an Oracle CQL Processor Cache Source](#)
- [Configuring an Oracle CQL Processor for Parallel Query Execution](#)
- [Handling Faults](#)
- [Example Oracle CQL Processor Configuration Files](#)

### Overview of Oracle CQL Processor Configuration

An Oracle Event Processing application contains one or more 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, such as a financial data feed or the Oracle Event Processing 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 Fusion Middleware CQL Language Reference for Oracle Event Processing*.

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 17-1](#), the corresponding

component configuration file `processor` element is shown in [Example 17-2](#).

**Example 17-1 EPN Assembly File Oracle CQL Processor Id: `proc`**

```
<wlevs:processor id="proc">
  <wlevs:table-source ref="Stock" />
</wlevs:processor>
```

**Example 17-2 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>
</processor>
```

You can create a `processor` element in any of the following component configuration files:

- The default Oracle Event Processing 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 Event Processing IDE for Eclipse creates one component configuration file and one EPN assembly file. When you create an Oracle CQL processor using Oracle Event Processing IDE for Eclipse, by default, the processor element is added to the default component configuration file `META-INF/wlevs/config.xml` file. Using Oracle Event Processing 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 Event Processing application bundle. You can later update this configuration at runtime using Oracle Event Processing Visualizer, the `wlevs.Admin` utility, or manipulating the appropriate JMX Mbeans directly.

For more information, see:

- [Section , "Overview of Component Configuration Files"](#)
- [Section , "Overview of EPN Assembly Files"](#)
- [Section , "Creating EPN Assembly Files"](#)
- *Oracle Fusion Middleware Visualizer User's Guide for Oracle Event Processing*
- "wlevs.Admin Command-Line Reference" in the *Oracle Fusion Middleware Administrator's Guide for Oracle Event Processing*

- "Configuring JMX for Oracle Event Processing" in the *Oracle Fusion Middleware Administrator's Guide for Oracle Event Processing*

For more information on Oracle CQL processor configuration, see:

- [Section , "Controlling Which Queries Output to a Downstream Channel"](#)
- [Section , "Configuring an Oracle CQL Processor"](#)
- [Section , "Configuring an Oracle CQL Processor Table Source"](#)
- [Section , "Configuring an Oracle CQL Processor Cache Source"](#)
- [Section , "Example Oracle CQL Processor Configuration Files"](#)

## Controlling Which Queries Output to a Downstream Channel

If you configure an Oracle CQL processor with more than one query, by default, all queries output their results to the downstream channel.

You can control which queries may output their results to a downstream channel using the `channel selector` element to specify a space delimited list of query names that may output their results on this channel.

You may configure a `channel` element with a `selector` before creating the queries in the upstream processor. In this case, you must specify query names that match the names in the `selector`.

For more information, see [Section , "Controlling Which Queries Output to a Downstream Channel: selector"](#).

## Configuring an Oracle CQL Processor

You can configure a processor manually or by using the Oracle Event Processing IDE for Eclipse.

See [Section , "Component Configuration Schema wlevs\\_application\\_config.xsd"](#) for the complete XSD Schema that describes the processor component configuration file.

See [Section , "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 , "How to Configure an Oracle CQL Processor Using Oracle Event Processing IDE for Eclipse"](#)
- [Section , "How to Create an Oracle CQL Processor Component Configuration File Manually"](#)

## How to Configure an Oracle CQL Processor Using Oracle Event Processing IDE for Eclipse

The most efficient and least error-prone way to create and edit a processor is to use the Oracle Event Processing IDE for Eclipse. Optionally, you can create and edit a processor manually (see [Section , "How to Create an Oracle CQL Processor Component Configuration File Manually"](#)).

**To configure an Oracle CQL processor using Oracle Event Processing IDE for Eclipse:**

1. Use Oracle Event Processing IDE for Eclipse to create a processor.

See [Section , "How to Create a Processor Node"](#).

When you use the EPN editor to create an Oracle CQL processor, Oracle Event Processing IDE for Eclipse prompts you to choose either the default component configuration file or a new component configuration file. For more information, see [Chapter 7, "Oracle Event Processing IDE for Eclipse and the Event Processing Network"](#).

2. Right-click the processor node and select **Go to Configuration Source**.

Oracle Event Processing IDE for Eclipse opens the appropriate component configuration file. The default processor component configuration is shown in [Example 17-3](#).

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 17-3 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 Fusion Middleware CQL Language Reference for Oracle Event Processing*.

4. Select **File > Save**.
5. Optionally, configure additional Oracle CQL processor features in the assembly file:
  - [Section , "Controlling Which Queries Output to a Downstream Channel"](#)
  - [Section , "Configuring an Oracle CQL Processor Table Source"](#)
  - [Section , "Configuring an Oracle CQL Processor Cache Source"](#)

## **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 Event Processing IDE for Eclipse (see [Section , "How to Configure an Oracle CQL Processor Using Oracle Event Processing 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 Fusion Middleware CQL Language Reference for Oracle Event Processing*.

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 Event Processing knows to which particular processor component in the EPN assembly file this processor configuration applies. See [Section , "Creating EPN Assembly Files"](#) 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 xmlns:n1="http://www.bea.com/ns/wlevs/config/application">
  <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>
      <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
        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 , "Controlling Which Queries Output to a Downstream Channel"](#)
- [Section , "Configuring an Oracle CQL Processor Table Source"](#)
- [Section , "Configuring an Oracle CQL Processor Cache Source"](#)

## Configuring an Oracle CQL Processor Table Source

You can access a relational database table from an Oracle CQL query using:

- **table source:** using a table source, you may join a stream only with a `NOW` window and only to a single database table.

---

**Note:** Because changes in the table source are not coordinated in time with stream data, you may only join the table source to an event stream using a `Now` window and you may only join to a single database table. For more information, see "S[now]" in the *Oracle Fusion Middleware CQL Language Reference for Oracle Event Processing*.

To integrate arbitrarily complex SQL queries and multiple tables with your Oracle CQL queries, consider using the Oracle JDBC data cartridge instead.

---

For more information, [Section , "Configuring an Oracle CQL Processor Table Source"](#).

- **Oracle JDBC data cartridge:** using the Oracle JDBC data cartridge, you may integrate arbitrarily complex SQL queries and multiple tables and datasources with your Oracle CQL queries.

---

**Note:** Oracle recommends that you use the Oracle JDBC data cartridge to access relational database tables from an Oracle CQL statement.

---

For more information, see "Understanding the Oracle JDBC Data Cartridge" in the *Oracle Fusion Middleware CQL Language Reference for Oracle Event Processing*.

In all cases, you must define datasources in the Oracle Event Processing server `config.xml` file. For more information, see "Configuring Access to a Relational Database" in the *Oracle Fusion Middleware Administrator's Guide for Oracle Event Processing*.

Oracle Event Processing relational database table event sources are pull data sources: that is, Oracle Event Processing will periodically poll the event source.

In this section, assume that you create the table you want to access using the SQL statement that [Example 17-4](#) shows.

### **Example 17-4 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:

**Example 17–5 Oracle CQL Query on Relational Database Table Stock**

```
SELECT ExchangeStream.symbol, ExchangeStream.price, Stock.exchange
FROM   ExchangeStream [Now], Stock
WHERE  ExchangeStream.symbol = Stock.symbol
```

---

**Note:** Because changes in the table source are not coordinated in time with stream data, you may only join the table source to an event stream using a Now window and you may only join to a single database table.

To integrate arbitrarily complex SQL queries and multiple tables with your Oracle CQL queries, consider using the Oracle JDBC data cartridge instead.

For more information, see "Understanding the Oracle JDBC Data Cartridge" in the *Oracle Fusion Middleware CQL Language Reference for Oracle Event Processing*.

---

## How to Configure an Oracle CQL Processor Table Source Using Oracle Event Processing 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 Event Processing IDE for Eclipse.

### To configure an Oracle CQL processor table source using Oracle Event Processing IDE for Eclipse:

1. Create a data source for the database that contains the table you want to use.

[Example 17–6](#) shows an example Oracle Event Processing server `config.xml` file with data source `StockDS`.

### Example 17–6 Oracle Event Processing 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>
...
</nl:config>

```

For more information, see "Configuring Access to a Relational Database" in the *Oracle Fusion Middleware Administrator's Guide for Oracle Event Processing*.

2. Use Oracle Event Processing IDE for Eclipse to create a table node.  
See [Section , "How to Create a Basic Node"](#).
3. Use Oracle Event Processing IDE for Eclipse to create an Oracle CQL processor.  
See [Section , "How to Create a Processor Node"](#).
4. Connect the table node to the Oracle CQL processor node.  
See [Section , "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 Event Processing IDE for Eclipse opens the EPN assembly file for this table node.
6. Edit the table element as [Example 17-7](#) shows and configure the `table` element attributes as shown in [Table 17-1](#).

**Example 17-7 EPN Assembly File table Element**

```
<wlevs:table id="Stock" event-type="StockEvent" data-source="StockDs" />
```

**Table 17-1 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 <code>type-name</code> you specify for the table <code>event-type</code> you create in step 9.
data-source	The <code>data-source</code> name you specified in the Oracle Event Processing server <code>config.xml</code> 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 Event Processing 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 17-8](#) shows.

Set the `ref` attribute to the `id` of the table element you specified in step 6.

**Example 17–8 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 17–9](#) shows.

Create a property child element for each column of the table you want to access and configure the property attributes as described in in [Section , "Constraints on Event Types for Use With a Database Table Source"](#).

**Example 17–9 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="char[]" length="16" />
      <wlevs:property name="exchange" type="char[]" length="16" />
    </wlevs:properties>
  </wlevs:event-type>
  ...
</wlevs:event-type-repository>
```

For more information on creating event types, see:

- [Chapter , "Creating Event Types"](#)
- [Section , "Choosing a Data Type for an Event Type"](#)

10. Right-click the Oracle CQL processor node connected to the table in your EPN and select **Go to Configuration Source**.

Oracle Event Processing IDE for Eclipse opens the component configuration file for this Oracle CQL processor.

11. Edit the component configuration file to add Oracle CQL queries that use the table's `event-type` as shown in [Example 17–10](#).

**Example 17–10 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>
```

---

**Note:** Because changes in the table source are not coordinated in time with stream data, you may only use a `Now` window. For more information, see "S[Now]" in the *Oracle Fusion Middleware CQL Language Reference for Oracle Event Processing*.

---

## Configuring an Oracle CQL Processor Cache Source

You can configure an Oracle CQL processor to access the Oracle Event Processing cache.

For more information, see:

- [Section , "Overview of Integrating a Cache"](#)
- [Section , "Accessing a Cache From an Oracle CQL User-Defined Function"](#)
- [Section , "Accessing a Cache from an Oracle CQL Statement"](#)

## Configuring an Oracle CQL Processor for Parallel Query Execution

For improved performance, you can enable a CQL query to execute in parallel rather than serially, as it does by default. When the CQL code supports it, you can configure a query so that it can process incoming events in parallel when multiple threads are available to the CQL processor.

You should enable parallel query execution only in cases where the relative order of the query output events is unimportant to the query's downstream client. For example, event ordering probably isn't important if your query is intended primarily to filter events, such as to deliver to clients a set of stock transactions involving a particular company, where the transaction sequence is irrelevant.

By default (without enabling parallel execution), queries process events from a channel serially. For events routed through a channel that uses a system timestamp, event order is the order in which events are received; through a channel that is application timestamped, event order is the order determined by a timestamp value included in the event. Relaxing the total order constraint allows the configured query to not consider event order for that query, processing events in parallel where possible.

## Setting Up Parallel Query Execution Support

While specifying support for parallel query execution is at its core a simple configuration task, be sure to follow the other steps below so that you get the most out of the feature.

- Use the `ordering-constraint` attribute to support parallel execution.
- Make sure you have enough threads calling into the processor to meet your performance goals. The maximum amount of parallel query execution is constrained by the number of threads available to the CQL processor. For example, if an adapter upstream of the processor supports the number of threads you need and there is a channel between the adapter and the processor, try configuring the channel with a `max-threads` count of 0 so that it acts as a pass-through.

If you don't want a pass-through, be sure to configure the query's upstream channel with a `max-threads` value greater than 1. (To make a `max-threads` value setting useful, you'll need to also set the `max-size` attribute to a value greater than 0.) For more information, see [Chapter 10, "Connecting EPN Stages Using Channels"](#).

- Follow other guidelines related to setting the `max-threads` attribute value. For example, to make a `max-threads` value setting useful, you'll need to also set the `max-size` attribute to a value greater than 0.
- Ensure, if necessary, that a bean receiving the query results is thread-aware, such as by using synchronized blocks. For example, you might need to do so if the

bean's code builds a list from results received from queries executed on multiple threads.

## Using the `ordering-constraint` Attribute

You enable parallel query execution by relaxing the default ordering constraint that ensures that events are processed serially. You do this by setting the `ordering-constraint` attribute on a query or view element.

In [Example 17-11](#), the `ordering-constraint` attribute is set to `UNORDERED` so that the query will execute in parallel whenever possible:

### **Example 17-11 Query Configured to Allow Parallel Execution**

```
<query id="myquery" ordering-constraint="UNORDERED">
  SELECT symbol FROM S WHERE price > 10
</query>
```

The `ordering-constraint` attribute supports the following three values:

- `ORDERED` means that the order of output events (as implied by the order of input events) is important. The CQL engine will process events serially. This is the default behavior.
- `UNORDERED` means that order of the output events is *not* important to the consumer of the output events. This gives the freedom to the CQLProcessor to process events in parallel on multiple threads. When possible, the query will execute in parallel on multiple threads to process the events.
- `PARTITION_ORDERED` means that you're specifying that order of output events within a partition is to be preserved (as implied by the order of input events) while order of output events across different partitions is *not* important to the consumer of the output events. This relaxation provides some freedom to the CQL engine to process events across partitions in parallel (when possible) on multiple threads.

Use the `PARTITION_ORDERED` value when you want to specify that events conforming to a given partition are processed serially, but that order can be disregarded across partitions and events belonging to different partitions may be processed in parallel. When using the `PARTITION_ORDERED` value, you must also add the `partition-expression` attribute to specify which expression for partitioning should be the basis for relaxing the cross-partition ordering constraint.

In [Example 17-12](#), the `GROUP BY` clause partitions the output based on symbol values. The `partition-expression` attribute specifies that events in a given subset of events corresponding to a particular symbol value should be handled serially. Across partitions, on the other hand, order can be disregarded.

### **Example 17-12 Query Configured to Allow Parallel Execution Across Partitions**

```
<query id="myquery" ordering-constraint="PARTITION_ORDERED"
  partitioning-expression="symbol">
  SELECT
    COUNT(*) as c, symbol
  FROM
    S[RANGE 1 minute]
  GROUP BY
    symbol
</query>
```

## Using partition-order-capacity with Partitioning Queries

In general, you'll probably see improved performance for queries by making more threads available and setting the `ordering-constraint` attribute so that they're able to execute in parallel when possible. As with most performance tuning techniques, a little trial and error with these settings should yield a combination that gets better results.

However, in some cases where your queries use partitioning -- and you've set the `ordering-constraint` attribute to `PARTITION_ORDERED` -- you might not see the amount of scaling you'd expect. For example, consider a case in which running with four threads doesn't improve performance very much over running with two threads. In such a case, you can try using the `partition-order-capacity` value to get the most out of CQL engine characteristics at work with queries that include partitions.

The `partition-order-capacity` value specifies the maximum amount of parallelism that will be permitted within a given processor instance when processing a `PARTITION_ORDERED` query. When available threads are handling events belonging to different partitions, the value sets a maximum number of threads that will be allowed to simultaneously run in the query.

As with other aspects of performance tuning, getting the most out of `partition-order-capacity` may take a bit of experimentation. When tuning with `partition-order-capacity`, a good starting point is to set it equal to the maximum number of threads you expect to have active in any CQL processor instance. In some cases (for example, at high data rates or with expensive processing downstream from the CQL processor), it may be helpful to set the `partition-order-capacity` value even higher than the available number of threads. However, you should only do this if performance testing confirms that it's helpful for a given application and load.

The `partition-order-capacity` value is set from one of four places, two of which are fallbacks when you don't explicitly set it yourself. These are, in order of precedence:

1. The `partition-order-capacity` element set on a channel configuration. If you specify this on the input channel for a processor, it takes effect for any `PARTITION_ORDERED` queries in that processor. For more information, see [Section , "partition-order-capacity" in Appendix D, "Schema Reference: Component Configuration wlevs\\_application\\_config.xsd"](#).
2. The `partition-order-capacity` property in server configuration. This value will be used for all `PARTITION_ORDERED` queries running on the server unless the value is set on a channel. For more information, see [Section , "partition-order-capacity" in Appendix F, "Schema Reference: Server Configuration wlevs\\_server\\_config.xsd"](#).
3. The `max-threads` value set on a channel configuration. If you specify this on the input channel for a processor, it takes effect for any `PARTITION_ORDERED` queries in that processor
4. A system default value (currently set to 4) is used if you don't specify either a `partition-order-capacity` value or `max-threads` value, or if the `max-threads` value is set to 0 (meaning it's a pass-through channel).

When using `partition-order-capacity`, keep in mind the following:

- The `partition-order-capacity` value is only useful when you're setting the `ordering-constraint` attribute to `PARTITION_ORDERED`.
- Increasing `partition-order-capacity` generally increases parallelism and scaling. For example, if your profiling reveals lock contention bottlenecks, you might find it helpful to increase `partition-order-capacity` to see if contention is reduced.

- Setting `partition-order-capacity` even higher than the number of available threads can be helpful in some cases because of the particular way partitioning is done in the CQL processor.
- There is some resource cost in memory used by specifying very high values.
- Tuning this parameter is very dependent on details of the application and the input rate. Tuning by experimentation may be necessary to determine an optimal value.

## Limitations

Think of parallel query execution as a performance enhancement feature that you specify support for so that the CQL processor can use it *whenever possible*. Not all queries can be executed in parallel. This includes queries using certain CQL language features.

For example, if your query uses some form of aggregation -- such as to find the maximum value from a range of values -- the CQL processor may not be able to fully execute the query in parallel (this is needed to guarantee the correct result considering the ordering constraint). Some query semantics in themselves also constrain the query to ordered processing. Such queries will be executed serially regardless of whether you specify support for parallel execution.

Also, the `IStream`, `RStream` and `DStream` operators maintain the state of their operand for processing, making it necessary for the CQL processor to synchronize threads in order to execute the query.

Note that the CQL processor always respects the semantic intention of your query. In cases where the `ordering-constraint` attribute would change this intention, the attribute is coerced to a value that keeps the intention intact.

If you're using the `partitioning-expression` attribute, keep in mind that the attribute supports a single expression only. Entering multiple property names for the value is not supported.

## Handling Faults

You can write code to handle faults that occur in code that does not have an inherent fault handling mechanism. This includes Oracle CQL code and multi-threaded EPN channels. By default, the CQL language has no mechanism for handling errors that occur, as does Java with its `try/catch` structure. To handle faults that occur in CQL, you can write a fault handler, then connect the handler to the EPN stage for which it handles faults, such as an Oracle CQL processor.

You can also associate a fault handler with a multi-threaded channel -- that is, a channel whose `max-threads` setting is greater than 0. This provides fault handling in the case of exceptions that are thrown to the channel from a stage that is downstream of the channel. Note that channels whose `max-threads` setting is 0 are pass-through channels that already re-throw exception to their upstream stages. For additional information specific to fault handlers for channels, see [Section , "Handling Faults in Channels"](#).

A fault handler is a Java class that implements the `com.bea.wlevs.ede.api.FaultHandler` interface. You connect the class to an EPN stage by registering your fault handler as an OSGi service and associating it with the stage. For more information about OSGi, see [Appendix A, "Additional Information about Spring and OSGi"](#).



Without a custom fault handler, you get the following default fault handling behavior:

- When an exception occurs in Oracle CQL, the CQL engine catches the exception and stops the query processor.
- If an exception occurs in a stage that is downstream of the processor, then that stage will be dropped as a listener.
- Exceptions are logged (under the CQLServer category) and the events that are part of the exception's cause are discarded.
- Upstream stages are not notified of the failure.

When using custom fault handlers you write, you can:

- Associate a fault handler with an Oracle CQL processor or multi-threaded channel so that faults in those stages are thrown as exceptions to the handler. There, you can handle or re-throw the exception.
- Allow query processing to continue as your code either handles the exception or re-throws it to the stage that is next upstream.
- Save event data from being lost while handling a fault. For example, if you have configured a connection to a data source, you could save event data there.
- Log fault and event information when faults occur.
- Use multiple fault handlers where needed in an EPN so that exceptions thrown upstream will be handled when they reach other Oracle CQL processors and channels.

In other words, consider associating a fault handler with a stage that does not have its own mechanism for responding to faults, including Oracle CQL processors and multi-threaded channels. Other stages, such as custom adapters you write in Java, which have their own exception-handling model, would not benefit from a fault handler.

Queries can continue as your fault handling code evaluates the fault to determine what action should be taken, including re-throwing the fault to a stage that is upstream of the CQL processor.

For example, the upstream stage of the CQL processor could be the JMS subscriber adapter, which has the option of rolling back the JMS transaction (if the session is transacted), allowing the event to be re-delivered. It can also commit the transaction if the event has been re-delivered already and found that the problem is not solvable.

Note that even when you are using a custom fault handler, query state is reset after a fault as if the query had been stopped and restarted. Yet contrast this with the default behavior, where the query is stopped and all subsequent events are dropped.

## Implementing a Fault Handler Class

You create a fault handler class by implementing the `com.bea.wlevs.ede.api.FaultHandler` interface. After you have written the class, you associated it with the stage for which it will handle faults by registering it as an OSGi service. For more information, see [Section , "Registering a Fault Handler"](#).

Your implementation of the interface's one method, `handleFault`, receives exceptions for the EPN stage with which the handler is associated. The exception itself is either an instance of `com.bea.wlevs.ede.api.EventProcessingException` or, if there has been a JVM error, an instance of `java.lang.Error`.

The method also receives a string array containing the names of upstream stages, or *catchers*, to which the exception will go if your code re-throws it. If there is more than one catcher in the array, your re-thrown exception will go to all of them. There are two cases when the catchers array will be empty: when the exception occurs while executing a temporal query and if the exception is thrown to a channel's fault handler. In these cases, the fault handler is executed in the context of a background thread; there is no linkage to upstream stages.

An exception that is re-thrown from a fault handler will travel back up through upstream EPN stages until it is either caught or reaches a stage that cannot catch it (such as a processor or multi-threaded channel that does not have an associated fault handler). Note that if you re-throw an exception, any channels in the catchers list must have an associated fault handler in order to catch the exception.

The `EventProcessingException` instance could also be one of the exception types that extend that class, including `CQLExecutionException`, `ArithmeticExecutionException`, and others (be sure to see the *Oracle Fusion Middleware Java API Reference for Oracle Event Processing*). The `EventProcessingException` instance provides methods with which your code can retrieve insert, delete, and update events that were involved in generating the fault.

Your implementation of the method should do one of the following:

- Consume the fault in the way that a Java try/catch statement might. If your implementation does not re-throw the fault, then event processing will continue with subsequent events. However, query processing continues with its state reset as if the query had been restarted. Processing state is lost and processing begins fresh with events that follow those that provoked the fault.
- Re-throw the fault so that it will be received by upstream stages (or their fault handlers). As when the fault is consumed, queries continue processing events, although query state is reset with subsequent events. The upstream stage receiving the fault always has the option of explicitly stopping the offending query by using the CQL processor's MBean interface.

In [Example 17-13, "Fault Handler Class"](#), the code provides a high level illustration of handling a fault.

#### **Example 17-13 Fault Handler Class**

```
package com.example.faulthandler;

import com.bea.wlevs.ede.api.FaultHandler;

public class SimpleFaultHandler implements FaultHandler
{
    private String suppress;

    // Called by the server to pass in fault information.
    @Override
    public void handleFault(Throwable fault, String[] catchers) throws Throwable
    {
        // Log the fault.
        return;
    }
}
```

## Registering a Fault Handler

After you have written a fault handling class, you can associate it with an EPN stage by registering it as an OSGi service. The simplest way to do this is to register the handler declaratively in the EPN assembly file.

---

**Note:** Due to inherent OSGi behavior, runtime fault handler registration from your configuration happens asynchronously, meaning that a small amount of warm-up time might be required before the handler is able to receive faults. To be sure your handler is ready for the first events entering the network, consider adding a wait period before the application begins receiving events.

---

In [Example 17–14, "Code to Register a Fault Handler with an EPN Stage"](#), the EPN assembly file excerpt shows a `service` element stanza that registers the `SimpleFaultHandler` class as the fault handler for the Oracle CQL processor whose `id` is `exampleProcessor`.

### Example 17–14 Code to Register a Fault Handler with an EPN Stage

```
<osgi:service interface="com.bea.wlevs.ede.api.FaultHandler">
  <osgi:service-properties>
    <entry key="application.identity" value="myapp" />
    <entry key="stage.identity" value="exampleProcessor" />
  </osgi:service-properties>
  <bean class="com.example.faulthandler.SimpleFaultHandler" />
</osgi:service>

<!-- A processor with a user-defined function. -->
<wlevs:processor id="exampleProcessor" >
  ...
</wlevs:processor>
```

For more on the schema for registering OSGi services, see <http://static.springsource.org/osgi/docs/1.1.x/reference/html/appendix-schema.html>. For more on OSGi, see <http://en.wikipedia.org/wiki/OSGi>.

## Example Oracle CQL Processor Configuration Files

This section provides example Oracle CQL processor configuration files, including:

- [Section , "Oracle CQL Processor Component Configuration File"](#)
- [Section , "Oracle CQL Processor EPN Assembly File"](#)

### 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"><![CDATA[
            select mod(price)
            from filteredStream[partition by srcId, cusip rows 1]
        ]]></view>
        <query id="q1"><![CDATA[
            SELECT *
            FROM   lastEvents
            WHERE  price > 10000
        ]]></query>
    </rules>
</processor>
</nl: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 , "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 , "How to Configure a System-Timestamped Channel Using Oracle Event Processing IDE for Eclipse"](#)
- [Section , "How to Configure an Application-Timestamped Channel Using Oracle Event Processing IDE for Eclipse"](#)

## Oracle CQL Processor EPN Assembly File

The following example shows an EPN assembly file for an Oracle CQL processor.

```

<?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_1_1_6.xsd">

    <wlevs:event-type-repository>
        <wlevs:event-type type-name="ExchangeEvent">
            <wlevs:properties>
                <wlevs:property name="symbol" type="char[]" 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="char[]" length="16" />
                <wlevs:property name="price" type="java.lang.Double" />
            </wlevs:properties>
        </wlevs:event-type>
    </wlevs:event-type-repository>

```

```
        <wlevs:property name="exchange" type="char[]" length="16" />
    </wlevs:properties>
</wlevs:event-type>
<wlevs:event-type type-name="StockEvent">
    <wlevs:properties>
        <wlevs:property name="symbol" type="char[]" length="16" />
        <wlevs:property name="exchange" type="char[]" length="16" />
    </wlevs:properties>
</wlevs:event-type>
</wlevs:event-type-repository>

<!-- 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>
```



---

---

## Configuring Applications With Data Cartridges

This chapter describes how to configure the Oracle JDBC cartridge and Oracle Spatial cartridge, which extend Oracle Continuous Query Language (CQL), for use with Oracle Event Processing.

This chapter includes the following sections:

- [Understanding Data Cartridge Application Context](#)
- [How to Configure Oracle Spatial Application Context](#)
- [How to Configure Oracle JDBC Data Cartridge Application Context](#)

For more information on data cartridges, see "Introduction to Data Cartridges" in the *Oracle Fusion Middleware CQL Language Reference for Oracle Event Processing*.

### Understanding Data Cartridge Application Context

Depending on the data cartridge implementation, you may be able to define an application context that the Oracle Event Processing server propagates to an instance of the data cartridge and the complex objects it provides.

You may configure an application context for the following data cartridges:

- [Section , "How to Configure Oracle Spatial Application Context"](#)
- [Section , "How to Configure Oracle JDBC Data Cartridge Application Context"](#)

For more information on data cartridges, see "Introduction to Data Cartridges" in the *Oracle Fusion Middleware CQL Language Reference for Oracle Event Processing*.

### How to Configure Oracle Spatial Application Context

You define an application context for an instance of Oracle Spatial using element `spatial:context` in your Oracle Event Processing application's Event Processing Network (EPN) assembly file.

All constructors and methods from `com.oracle.cartridge.spatial.Geometry` and Oracle Spatial functions are aware of `spatial:context`. For example, the SRID is automatically set from the value in the Oracle Spatial application context.

For more information, see:

- "SDO\_SRID" in the *Oracle Spatial Developer's Guide* at [http://download.oracle.com/docs/cd/E11882\\_01/appdev.112/e11830/sdo\\_objrelschem.htm#SPATL492](http://download.oracle.com/docs/cd/E11882_01/appdev.112/e11830/sdo_objrelschem.htm#SPATL492)

- "Understanding Oracle Spatial" in the *Oracle Fusion Middleware CQL Language Reference for Oracle Event Processing*

**To configure Oracle Spatial application context:**

1. Open the EPN editor in the Oracle Event Processing IDE for Eclipse.  
See [Section , "Opening the EPN Editor"](#).
2. Import the package `com.oracle.cep.cartridge.spatial` into your Oracle Event Processing application's `MANIFEST.MF` file.  
For more information, see [Section , "How to Import a Package"](#).
3. Right-click any component and select **Go to Assembly Source**.
4. Edit the EPN file to add the required namespace and schema location entries as [Example 18–1](#) shows:

**Example 18–1 EPN Assembly File: Oracle Spatial Namespace and Schema Location**

```
<?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:spatial="http://www.oracle.com/ns/ocep/spatial"
       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_1_1_6.xsd"
http://www.oracle.com/ns/ocep/spatial
http://www.oracle.com/ns/ocep/spatial/ocep-spatial.xsd">
```

5. Edit the EPN file to add a `spatial:context` element as [Example 18–2](#) shows.

**Example 18–2 spatial:context Element in EPN Assembly File**

```
<spatial:context id="SpatialGRS80" />
```

6. Assign a value to the `id` attribute that is unique in this EPN.  
This is the name you will use to reference this application context in subsequent Oracle CQL queries. In [Example 18–2](#), the `id` is `SpatialGRS80`.

---

**Note:** The `id` value must not equal the Oracle Spatial name `spatial`. For more information, see "Data Cartridge Name" in the *Oracle Fusion Middleware CQL Language Reference for Oracle Event Processing*.

---

7. Configure the other attributes of the `spatial:context` element to suit your application requirements.

[Table 18–1](#) lists the attributes of the `spatial:context` element.

**Table 18–1 spatial:context Element Attributes**

Attribute	Description
<code>anyinteract-tolerance</code>	The default tolerance for contain or inside operator. Default: 0.0000005



**Table 18–1 (Cont.) spatial:context Element Attributes**

Attribute	Description
rof	Defines the Reciprocal Of Flattening (ROF) parameter used for buffering and projection. Default: 298.257223563
sma	Defines the Semi-Major Axis (SMA) parameter used for buffering and projection. Default: 6378137.0
srid	SRID integer. Valid values are: <ul style="list-style-type: none"> <li>■ CARTESIAN: for cartesian coordinate system.</li> <li>■ LAT_LNG_WGS84_SRID: for WGS84 coordinate system.</li> <li>■ An integer value from the Oracle Spatial SDO_COORD_SYS table COORD_SYS_ID column.</li> </ul> Default : LAT_LNG_WGS84_SRID
tolerance	The minimum distance to be ignored in geometric operations including buffering. Default: 0.000000001

[Example 18–3](#) shows how to create a spatial context named `SpatialGRS80` in an EPN assembly file using the Geodetic Reference System 1980 (GRS80) coordinate system (`srid="4269"`).

**Example 18–3 spatial:context Element in EPN Assembly File**

```
<spatial:context id="SpatialGRS80" srid="4269" sma="63787.0" rof="298.25722101" />
```

**8. Create Oracle CQL queries that reference this application context by name.**

[Example 18–4](#) shows how to reference a `spatial:context` in an Oracle CQL query. In this case, the query uses link name `SpatialGRS80` (defined in [Example 18–2](#)) to propagate this application context to the Oracle Spatial. The `spatial:context` attribute settings of `SpatialGRS80` are applied to the `createPoint` method call. Because the application context defines the SRID, you do not need to pass that argument into the `createPoint` method.

**Example 18–4 Referencing spatial:context in an Oracle CQL Query**

```
<view id="createPoint">
  select com.oracle.cep.cartridge.spatial.Geometry.createPoint@SpatialGRS80(lng,
  lat, 0d)
  from CustomerPos[NOW]
</view>
```

For more information, see "Using Oracle Spatial" in the *Oracle Fusion Middleware CQL Language Reference for Oracle Event Processing*.

## How to Configure Oracle JDBC Data Cartridge Application Context

You define an application context for an instance of an Oracle JDBC data cartridge using:

- A `jdbc:jdbc-context` element in the EPN assembly file.
- A `jc:jdbc-ctx` element in the component configuration file.

The `jc:jdbc-ctx` element:

- references one and only one `jdbc:jdbc-context`
- references one and only one `data-source`
- defines one or more SQL functions

---



---

**Note:** You must provide alias names for every `SELECT` list column in the SQL function.

---



---

For more information see, "Understanding the Oracle JDBC Data Cartridge" in the *Oracle Fusion Middleware CQL Language Reference for Oracle Event Processing*.

**To configure Oracle JDBC data cartridge application context:**

1. Open the EPN editor in the Oracle Event Processing IDE for Eclipse.  
See [Section , "Opening the EPN Editor"](#).
2. Right-click any component and select **Go to Assembly Source**.
3. Edit the EPN file to add the required namespace and schema location entries as [Example 18-5](#) shows:

**Example 18-5 EPN Assembly File: Oracle JDBC Data Cartridge Namespace and Schema Location**

```
<?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:jdbc="http://www.oracle.com/ns/ocep/jdbc"
       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_1_1_6.xsd
http://www.oracle.com/ns/ocep/jdbc
http://www.oracle.com/ns/ocep/jdbc/ocep-jdbc.xsd">
```

4. Edit the EPN file to add a `jdbc:jdbc-context` element as [Example 18-6](#) shows.

**Example 18-6 jdbc:jdbc-context Element in EPN Assembly File: id**

```
<jdbc:jdbc-context id="JdbcCartridgeOne"/>
```

5. Assign a value to the `id` attribute that is unique in this EPN.

This is the name you will use to reference this application context in subsequent Oracle CQL queries. In [Example 18-6](#), the `id` is `JdbcCartridgeOne`.

---



---

**Note:** The `id` value must not equal the Oracle JDBC data cartridge name `jdbc`. For more information, see "Data Cartridge Name" in the *Oracle Fusion Middleware CQL Language Reference for Oracle Event Processing*.

---



---

6. Right-click the desired processor and select **Go to Configuration Source**.

7. Edit the component configuration file to add the required namespace entries as [Example 18–7](#) shows:

**Example 18–7 Component Configuration File: Oracle JDBC Data Cartridge Namespace**

```
<?xml version="1.0" encoding="UTF-8"?>
<wlevs:config
  xmlns:wlevs="http://www.bea.com/ns/wlevs/config/application"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns:jc="http://www.oracle.com/ns/ocep/config/jdbc"
  xsi:schemaLocation="
    http://www.oracle.com/ns/ocep/config/jdbc
    http://www.oracle.com/ns/ocep/config/jdbc/ocep_jdbc_context.xsd">
```

8. Edit the component configuration file to add a `jc:jdbc-ctx` element as [Example 18–8](#) shows.

**Example 18–8 jc:jdbc-ctx Element in Component Configuration File**

```
<jc:jdbc-ctx>
</jc:jdbc-ctx>
```

9. Add a name child element whose value is the name of the Oracle JDBC application context you defined in the EPN assembly file as [Example 18–9](#) shows.

**Example 18–9 jc:jdbc-ctx Element in Component Configuration File: name**

```
<jc:jdbc-ctx>
  <name>JdbcCartridgeOne</name>
</jc:jdbc-ctx>
```

10. Add a data-source child element whose value is the name of a datasource defined in the Oracle Event Processing server `config.xml` file.

For more information, see:

- "Configuring JDBC for Oracle Event Processing" in the *Oracle Fusion Middleware Administrator's Guide for Oracle Event Processing*.
- [Section , "data-source"](#)

[Example 18–10](#) shows how to specify the datasource named `StockDS`.

**Example 18–10 jc:jdbc-ctx Element in Component Configuration File: data-source**

```
<jc:jdbc-ctx>
  <name>JdbcCartridgeOne</name>
  <data-source>StockDS</data-source>
</jc:jdbc-ctx>
```

11. Create one or more SQL functions using the function child element as [Example 18–11](#) shows.

**Example 18–11 jc:jdbc-ctx Element in Component Configuration File: function**

```
<jc:jdbc-ctx>
  <name>JdbcCartridgeOne</name>
  <data-source>StockDS</data-source>
  <function name="getDetailsByOrderIdName">
    <param name="inpOrderId" type="int" />
    <param name="inpName" type="char" />
    <return-component-type>
      com.oracle.cep.example.jdbc_cartridge.RetEvent
    </return-component-type>
```

```

<sql><![CDATA[
    SELECT
        Employee.empName as employeeName,
        Employee.empEmail as employeeEmail,
        OrderDetails.description as description
    FROM
        PlacedOrders, OrderDetails , Employee
    WHERE
        PlacedOrders.empId = Employee.empId AND
        PlacedOrders.orderId = OrderDetails.orderId AND
        Employee.empName = :inpName AND
        PlacedOrders.orderId = :inpOrderId
]]></sql>
</function>
</jc:jdbc-ctx>

```

---

**Note:** You must provide alias names for every SELECT list column in the SQL query.

---

For more information, see "Defining SQL Statements" in the *Oracle Fusion Middleware CQL Language Reference for Oracle Event Processing*.

12. Create Oracle CQL queries that invoke the SQL functions using the Oracle JDBC data cartridge application context.

[Example 18–12](#) shows how to reference a `jdbc:jdbc-context` in an Oracle CQL query. In this case, the query uses link name `JdbcCartridgeOne` (defined in [Example 18–11](#)) to propagate this application context to the Oracle JDBC data cartridge. The Oracle CQL query in [Example 18–12](#) invokes the function `getDetailsByOrderIdName` defined by Oracle JDBC data cartridge context `JdbcCartridgeOne`.

**Example 18–12 Referencing JDBC Application Context in an Oracle CQL Query**

```

<processor>
  <name>Proc</name>
  <rules>
    <query id="q1"><![CDATA[
      RStream(
        select
          currentOrder.orderId,
          details.orderInfo.employeeName,
          details.orderInfo.employeeemail,
          details.orderInfo.description
        from
          OrderArrival[now] as currentOrder,
          TABLE(getDetailsByOrderIdName@JdbcCartridgeOne(
            currentOrder.orderId, currentOrder.empName
          ) as orderInfo
        ) as details
      )
    ]></query>
  </rules>
</processor>

```

For more information see, "Defining Oracle CQL Queries With the Oracle JDBC Data Cartridge" in the *Oracle Fusion Middleware CQL Language Reference for Oracle Event Processing*.

---

---

## Querying an Event Stream with Oracle EPL

This chapter describes how to configure Oracle Event Processing Language (EPL) processors for Oracle Event Processing event processing networks. EPL is deprecated; new applications should use Oracle Continuous Query Language.

This chapter includes the following sections:

- [Overview of EPL Processor Component Configuration](#)
- [Configuring an EPL Processor](#)
- [Configuring an EPL Processor Cache Source](#)
- [Example EPL Processor Configuration Files](#)

---

---

**Note:** Oracle CQL replaces Event Processing Language (EPL) in Oracle Event Processing 11g Release 1 (11.1.1). Oracle Event Processing supports EPL for backwards compatibility. For more information, see [Chapter 17, "Querying an Event Stream with Oracle CQL"](#).

---

---

### Overview of EPL Processor Component Configuration

An Oracle Event Processing application contains one or more 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 Event Processing 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 Fusion Middleware EPL Language Reference for Oracle Event Processing*.

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 Event Processing 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 19-1](#), the corresponding component configuration file processor element is shown in [Example 19-2](#).

**Example 19-1 EPN Assembly File EPL Processor Id: proc**

```
<wlevs:processor id="proc" provider="ep1" >
  <wlevs:table-source ref="Stock" />
</wlevs:processor>
```

**Example 19-2 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 Event Processing 11g Release 1 (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 `ep1` as [Example 19-1](#) shows. Oracle Event Processing supports EPL for backwards compatibility. For more information, see [Chapter 17, "Querying an Event Stream with Oracle CQL"](#).

---

You can create a processor element in any of the following component configuration files:

- The default Oracle Event Processing 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 Event Processing IDE for Eclipse creates one component configuration file and one EPN assembly file. When you create an EPL processor using Oracle Event Processing 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 Event Processing application bundle. You can later update this configuration at runtime using Oracle Event Processing Visualizer, the `wlevs.Admin` utility, or manipulating the appropriate JMX MBeans directly.

For more information, see:

- [Section , "Overview of Component Configuration Files"](#)
- [Section , "Overview of EPN Assembly Files"](#)
- [Section , "Creating EPN Assembly Files"](#)
- *Oracle Fusion Middleware Visualizer User's Guide for Oracle Event Processing*
- "wlevs.Admin Command-Line Reference" in the *Oracle Fusion Middleware Administrator's Guide for Oracle Event Processing*
- "Configuring JMX for Oracle Event Processing" in the *Oracle Fusion Middleware Administrator's Guide for Oracle Event Processing*

For more information on EPL processor configuration, see:

- [Section , "Configuring an EPL Processor"](#)
- [Section , "Configuring an EPL Processor Cache Source"](#)
- [Section , "Example EPL Processor Configuration Files"](#)

## 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 , "Component Configuration Schema wlevs\\_application\\_config.xsd"](#) for the complete XSD Schema that describes the processor configuration file.

See [Section , "Example EPL Processor Configuration Files"](#) for a complete example of a processor configuration file.

## 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 Fusion Middleware EPL Language Reference for Oracle Event Processing*.

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 Event Processing knows to which particular processor component in the EPN assembly file this processor configuration applies. See [Section , "Creating EPN Assembly Files"](#) 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 xmlns:n1="http://www.bea.com/ns/wlevs/config/application">
  <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="ep1"...>
  ...
</wlevs:processor>
<wlevs:processor id="secondProcessor" provider="ep1"...>
  ...
</wlevs:processor>
```

---



---

**Note:** Because Oracle CQL replaces Event Processing Language (EPL) in Oracle Event Processing 11g Release 1 (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 `ep1`. Oracle Event Processing supports EPL for backwards compatibility. For more information, see [Chapter 17, "Querying an Event Stream with Oracle CQL"](#).

---



---



---



---

**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 Fusion Middleware Administrator's Guide for Oracle Event Processing*.

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 , "Configuring an EPL Processor Cache Source"](#)

## Configuring an EPL Processor Cache Source

You can configure an EPL processor to access the Oracle Event Processing cache.

For more information, see:

- [Section , "Overview of Integrating a Cache"](#)
- [Section , "Accessing a Cache From an EPL User-Defined Function"](#)
- [Section , "Accessing a Cache From an EPL Statement"](#)

## Example EPL Processor Configuration Files

This section provides example Oracle CQL processor configuration files, including:

- [Section , "EPL Processor Component Configuration File"](#)
- [Section , "EPL Processor EPN Assembly File"](#)

### EPL Processor Component Configuration File

The following example shows how to configure one of the sample EPL queries shown in [Section , "Configuring an EPL Processor"](#) for the `myProcessor` EPL processor:

```
<?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="ep1" />` element to link these EPL rules with an actual `myProcessor` EPL processor instance (see [Section , "EPL Processor EPN Assembly File"](#)).

### EPL Processor EPN Assembly File

The following example shows an EPN assembly file for an EPL processor.

```
<?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_1_1_6.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" provider="ep1" />
<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>
```



## Configuring Event Record and Playback

This chapter describes how to configure event recording and playback for debugging Oracle Event Processing event processing networks, including how to specify an event persistence store and query the store.

This chapter includes the following sections:

- [Overview of Configuring Event Record and Playback](#)
- [Configuring Event Record and Playback in Your Application](#)
- [Creating a Custom Event Store Provider](#)

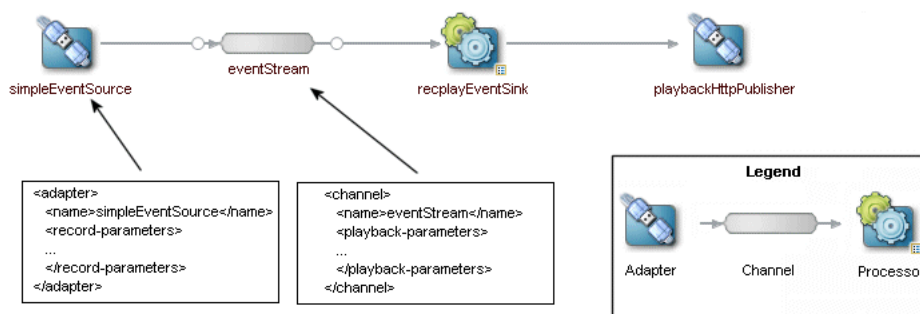
### Overview of Configuring Event Record and Playback

Oracle Event Processing 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 20–1** Configuring Record and Playback in an EPN



This section describes:

- [Section , "Storing Events in the Persistent Event Store"](#)
- [Section , "Recording Events"](#)
- [Section , "Playing Back Events"](#)
- [Section , "Querying Stored Events"](#)
- [Section , "Record and Playback Example"](#)

## Storing Events in the Persistent Event Store

When you record events, Oracle Event Processing server stores them in a persistent event store. You can use the persistent event store that Oracle Event Processing server provides or define your own:

- [Section , "Default Persistent Event Store"](#)
- [Section , "Custom Persistent Event Store"](#)
- [Section , "Persistent Event Store Schema"](#)

### Default Persistent Event Store

By default, Oracle Event Processing uses a Berkeley DB instance bundled with the Oracle Event Processing server to store recorded events.

Berkeley DB is a fast, scalable, transactional database engine with industrial grade reliability and availability. For more information, see:

- <http://www.oracle.com/technology/products/berkeley-db/je/index.html>
- <http://www.oracle.com/technology/documentation/berkeley-db/je/index.html>

By default, Oracle Event Processing server creates the Berkeley DB instance in:

```
ORACLE_CEP_HOME/user_projects/domains/domainname/servername/bdb
```

Where *ORACLE\_CEP\_HOME* refers to the directory in which you installed Oracle Event Processing (such as */oracle\_home*), *domainname* refers to the name of your domain, and *servername* refers to the name of your server (For example, */oracle\_cep/user\_projects/domains/mydomain/myserver*).

You can change this default by configuring the `bdb-config` element `db-env-path` child element as [Section , "Configuring an Event Store for Oracle Event Processing Server"](#) describes.

### Custom Persistent Event Store

Optionally, you can create a custom persistent event store provider to store recorded events. For example, you could specify a Relational Database Management System such as Oracle Database or Derby as your persistent event store.

For more information, see [Section , "Creating a Custom Event Store Provider."](#)

### Persistent Event Store Schema

You do not create the actual database schema used to store the recorded events. Oracle Event Processing server automatically does this for you after you deploy an application that uses the record and playback feature and recording begins.

For more information, see [Section , "Description of the Berkeley Database Schema"](#).

## 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 Event Processing stores the events that flow out of it to the appropriate store along with a timestamp of the time it was recorded.

## 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 Event Processing 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 Event Processing Visualizer or `wlevs.Admin`). Oracle Event Processing 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 Event Processing 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 Fusion Middleware Visualizer User's Guide for Oracle Event Processing*
- "Commands for Controlling Event Record and Playback" in the *Oracle Fusion Middleware Administrator's Guide for Oracle Event Processing*

## 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 default persistent event store provider included with Oracle Event Processing. You can also use these APIs to delete old events from the event store.

## 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\replay` directory, where `ORACLE_CEP_HOME` refers to the main Oracle Event Processing installation directory, such as `d:\oracle_cep`.

For details about running and building the example, see [Section , "Event Record and Playback Example"](#).

## 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.

### To configure record and playback of events in your application:

1. Optionally configure the Berkeley database event store for your Oracle Event Processing server instance.

You may use the default Berkeley database configuration as is. You only need to make configuration changes to customize the location of the Berkeley database instance or to tune performance.

See [Section , "Configuring an Event Store for Oracle Event Processing 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 , "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 , "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 Event Processing Visualizer or `wlevs.Admin` to start recording. You must always use these administration tools to start and end the playback of events.

See [Section , "Starting and Stopping the Record and Playback of Events."](#)

## Configuring an Event Store for Oracle Event Processing Server

You may use the default Berkeley database configuration as is. You only need to make configuration changes to customize the location of the Berkeley database instance or to tune performance.



For more information, see [Section , "Creating a Custom Event Store Provider"](#).

**To configure an event store for Oracle Event Processing server:**

1. Stop your Oracle Event Processing 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. Edit the `bdb-config` element to the `config.xml` file.

[Example 20–1](#) shows a fully configured `bdb-config` element.

**Example 20–1 *bdb-config* Element**

```
<bdb-config>
  <db-env-path>bdb</db-env-path>
  <cache-size>1000</cache-size>
</bdb-config>
```

[Table 20–1](#) lists the child elements of `bdb-config` that you can specify.

**Table 20–1 *Child Elements of bdb-config***

Child Element	Description
db-env-path	Specifies the subdirectory in which Oracle Event Processing server creates Berkeley database instances relative to the <code>DOMAIN_DIR/servername/config</code> directory of your server, where <code>DOMAIN_DIR</code> refers to the domain directory, such as <code>/oracle_cep/user_projects/domains/myDomain</code> and <code>servername</code> refers to the name of your server, such as <code>defaultserver</code> .  Default: <code>bdb</code>
cache-size	Specifies the amount of memory, in bytes, available for Berkeley database cache entries. You can adjust the cache size to tune Berkeley database performance.  For more information, see: <ul style="list-style-type: none"> <li>▪ <a href="http://www.oracle.com/technology/documentation/berkeley-db/je/GettingStartedGuide/cachesize.html">http://www.oracle.com/technology/documentation/berkeley-db/je/GettingStartedGuide/cachesize.html</a>.</li> <li>▪ <a href="http://www.oracle.com/technology/documentation/berkeley-db/je/java/com/sleepycat/je/EnvironmentMutableConfig.html#setCacheSize(long)">http://www.oracle.com/technology/documentation/berkeley-db/je/java/com/sleepycat/je/EnvironmentMutableConfig.html#setCacheSize(long)</a></li> </ul> Default: <code>je.maxMemoryPercent * JVM maximum memory</code>

4. Restart your Oracle Event Processing server instance.

## 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 , "Overview of 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 , "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 version="1.0" encoding="UTF-8"?>
  <nl:config xmlns:nl="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>
    ...
  </nl: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:

```
<adapter>
  <name>simpleEventSource</name>
  <record-parameters>
    <dataset-name>recplay_sample</dataset-name>
    <event-type-list>
      <event-type>SimpleEvent</event-type>
    </event-type-list>
    <batch-size>1</batch-size>
    <batch-time-out>10</batch-time-out>
  </record-parameters>
</adapter>
```

[Table 20–2](#) lists the child elements of `record-parameters` that you can specify. Only `dataset-name` is required.

**Table 20–2 Child Elements of record-parameters**

Child Element	Description
dataset-name	<p>Specifies the group of data that the user wants to group together. In the case of BDB provider, the dataset name will be used as the database environment in Berkeley database.</p> <p>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.</p> <p>When configuring the Oracle RDBMS-based provider, you are required to specify this element.</p>
event-type-list	<p>Specifies the event types that are recorded to the event store. If this element is not specified, then Oracle Event Processing records <i>all</i> event types that flow out of the component.</p> <p>Use the <code>event-type</code> child component to list one or more events, such as:</p> <pre>&lt;event-type-list&gt;   &lt;event-type&gt;EventOne&lt;/event-type&gt;   &lt;event-type&gt;EventTwo&lt;/event-type&gt; &lt;/event-type-list&gt;</pre> <p>When configuring the Oracle RDBMS-based provider, you are required to specify this element.</p>

**Table 20–2 (Cont.) Child Elements of record-parameters**

Child Element	Description
time-range	<p>Specifies the time period during which recording should take place using a start and end time.</p> <p>The time period is configured by using a <code>start</code> child element to specify a start time and an <code>end</code> child element to specify the end time.</p> <p>Express the start and end time as XML Schema <code>dateTime</code> values of the form:</p> <pre>yyyy-mm-ddThh:mm:ss</pre> <p>For example, to specify that recording should start on January 20, 2010, at 5:00am and end on January 20, 2010, at 6:00 pm, enter the following:</p> <pre>&lt;time-range&gt;   &lt;start&gt;2010-01-20T05:00:00&lt;/start&gt;   &lt;end&gt;2010-01-20T18:00:00&lt;/end&gt; &lt;/time-range&gt;</pre> <p>For complete details of the XML Schema <code>dateTime</code> format, see <a href="http://www.w3.org/TR/xmlschema-2/#dateTime-lexical-representation">http://www.w3.org/TR/xmlschema-2/#dateTime-lexical-representation</a>.</p> <p>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 Event Processing Visualizer or <code>wlevs.Admin</code>.</p> <p>You can specify <code>time-range</code> or <code>time-range-offset</code>, but not both.</p>
time-range-offset	<p>Specifies the time period during which recording should take place, using a start time and a duration.</p> <p>The time period is configured by using a <code>start</code> child element to specify a start time and <code>duration</code> child element to specify the amount of time after the start time that recording should stop.</p> <p>Express the start time as an XML Schema <code>dateTime</code> value of the form:</p> <pre>yyyy-mm-ddThh:mm:ss</pre> <p>Express the duration in the form:</p> <pre>hh:mm:ss</pre> <p>For example, to specify that recording should start on January 20, 2010, at 5:00am and continue for 3 hours, enter the following</p> <pre>&lt;time-range-offset&gt;   &lt;start&gt;2010-01-20T05:00:00&lt;/start&gt;   &lt;duration&gt;03:00:00&lt;/duration&gt; &lt;/time-range-offset&gt;</pre> <p>For complete details of the XML Schema <code>dateTime</code> format, see <a href="http://www.w3.org/TR/xmlschema-2/#dateTime-lexical-representation">http://www.w3.org/TR/xmlschema-2/#dateTime-lexical-representation</a>.</p> <p>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 Event Processing Visualizer or <code>wlevs.Admin</code>.</p> <p>You can specify <code>time-range</code> or <code>time-range-offset</code>, but not both.</p>
batch-size	<p>Specifies the number of events that Oracle Event Processing picks up in a single batch from the event buffer to write the event store.</p> <p>Default value is 1000.</p>
batch-time-out	<p>Specifies the number of seconds that Oracle Event Processing waits for the event buffer window to fill up with the <code>batch-size</code> number of events before writing to the event store.</p> <p>Default value is 60</p>

**Table 20–2 (Cont.) Child Elements of record-parameters**

Child Element	Description
max-size	If specified, Oracle Event Processing 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 Event Processing 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.

## 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 , "Overview of 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"?>
  <nl:config xmlns:nl="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>
    ...
  </nl: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>
  </playback-parameters>
</channel>
```

[Table 20–3](#) lists the child elements of playback-parameters that you can specify. Only dataset-name is required.

**Table 20–3 Child Elements of playback-parameters**

Child Element	Description
dataset-name	<p>Specifies the group of data that the user wants to group together.</p> <p>In the case of BDB provider, the dataset name will be used as the database environment in Berkeley database.</p> <p>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.</p> <p>When configuring the Oracle RDBMS-based provider, you are required to specify this element.</p>
event-type-list	<p>Specifies the event types that are played back from the event store. If this element is not specified, then Oracle Event Processing plays back <i>all</i> event types.</p> <p>Use the event-type child component to list one or more events, such as:</p> <pre>&lt;event-type-list&gt;   &lt;event-type&gt;EventOne&lt;/event-type&gt;   &lt;event-type&gt;EventTwo&lt;/event-type&gt; &lt;/event-type-list&gt;</pre> <p>When configuring the Oracle RDBMS-based provider, you are required to specify this element.</p>
time-range	<p>Specifies the time period during which play back should take place using a start and end time.</p> <p>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.</p> <p>Express the start and end time as XML Schema dateTime values of the form:</p> <pre>yyyy-mm-ddThh:mm:ss</pre> <p>For example, to specify that play back should start on January 20, 2010, at 5:00am and end on January 20, 2010, at 6:00 pm, enter the following:</p> <pre>&lt;time-range&gt;   &lt;start&gt;2010-01-20T05:00:00&lt;/start&gt;   &lt;end&gt;2010-01-20T18:00:00&lt;/end&gt; &lt;/time-range&gt;</pre> <p>For complete details of the XML Schema dateTime format, see <a href="http://www.w3.org/TR/xmlschema-2/#dateTime-lexical-representation">http://www.w3.org/TR/xmlschema-2/#dateTime-lexical-representation</a>.</p> <p>If you do not specify a time period, then no events are played back when the application is deployed and play back will only happen after you explicitly start it using Oracle Event Processing Visualizer or wlevs.Admin.</p> <p>You can specify time-range or time-range-offset, but not both.</p>

**Table 20–3 (Cont.) Child Elements of playback-parameters**

Child Element	Description
time-range-offset	<p>Specifies the time period during which play back should take place, using a start time and a duration.</p> <p>The time period is configured by using a <code>start</code> child element to specify a start time and <code>duration</code> child element to specify the amount of time after the start time that play back should stop.</p> <p>Express the start time as an XML Schema <code>dateTime</code> value of the form:  <code>yyyy-mm-ddThh:mm:ss</code></p> <p>Express the duration in the form:  <code>hh:mm:ss</code></p> <p>For example, to specify that play back should start on January 20, 2010, at 5:00am and continue for 3 hours, enter the following</p> <pre>&lt;time-range-offset&gt;   &lt;start&gt;2010-01-20T05:00:00&lt;/start&gt;   &lt;duration&gt;03:00:00&lt;/duration&gt; &lt;/time-range-offset&gt;</pre> <p>For complete details of the XML Schema <code>dateTime</code> format, see <a href="http://www.w3.org/TR/xmlschema-2/#dateTime-lexical-representation">http://www.w3.org/TR/xmlschema-2/#dateTime-lexical-representation</a>.</p> <p>If you do not specify a time period, then no events are played back when the application is deployed and play back will only happen after you explicitly start it using Oracle Event Processing Visualizer or <code>wlevs.Admin</code>.</p> <p>You can specify <code>time-range</code> or <code>time-range-offset</code>, but not both.</p>
playback-speed	<p>Specifies the playback speed as a positive float.</p> <p>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.</p>
repeat	<p>Specifies whether to playback events again after the playback of the specified time interval is over.</p> <p>Valid values are <code>true</code> and <code>false</code>. Default value is <code>false</code>. A value of <code>true</code> means that the repeat of playback continues an infinite number of times until it is deliberately stopped. <code>false</code> means that events will be played back only once.</p>
max-size	<p>If specified, Oracle Event Processing 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.</p> <p>Default value is 1024.</p>
max-threads	<p>If specified, Oracle Event Processing 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 <code>max-size</code> is 0.</p> <p>The default value is 1.</p>

## 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 Event Processing, 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>
  <start>2010-01-20T05:00:00</start>
```

```
<end>2010-01-20T18:00:00</end>
</time-range>
```

then recording will automatically start on January 20, 2010 at 5:00 am.

The only way to start the playback of events, however, is by using Oracle Event Processing 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 Fusion Middleware Visualizer User's Guide for Oracle Event Processing*
- "Commands for Controlling Event Record and Playback" in the *Oracle Fusion Middleware Administrator's Guide for Oracle Event Processing*

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 Event Processing" in the *Oracle Fusion Middleware Administrator's Guide for Oracle Event Processing*
- *Oracle Fusion Middleware Java API Reference for Oracle Event Processing*

## Description of the Berkeley Database Schema

When you configure a stage for event record and playback, you specify a `dataset-name` to identify the recorded data.

Oracle Event Processing server creates a subdirectory with this name below the `db-env-path` you specify in your `bdb-config` element.

For example, consider the `bdb-config` element is as [Example 20–2](#) shows.

### **Example 20–2** Default `bdb-config` Element

```
<bdb-config>
  <db-env-path>bdb</db-env-path>
</bdb-config>
```

If your `dataset-name` is `test1`, then Oracle Event Processing server stores recorded data in directory:

```
ORACLE_CEP_HOME/user_projects/domains/domainname/servername/bdb/test1
```

Where `ORACLE_CEP_HOME` refers to the directory in which you installed Oracle Event Processing (such as `/oracle_home`), `domainname` refers to the name of your domain, and `servername` refers to the name of your server (For example, `/oracle_cep/user_projects/domains/mydomain/myserver`).

Within the `data-set` subdirectory, Oracle Event Processing creates a Berkeley database environment that contains a separate database for each event type you record. The database name is the same as the event type name as specified in the event type repository.

The database key is record time plus sequence number.

## Creating a Custom Event Store Provider

Oracle Event Processing 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 Event Processing 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 Fusion Middleware Java API Reference for Oracle Event Processing*.



---

---

## Testing Applications With the Load Generator and csvgen Adapter

This chapter describes how to use the included load generator and csvgen adapter to test Oracle Event Processing applications.

This chapter includes the following sections:

- [Overview of Testing Applications With the Load Generator and csvgen Adapter](#)
- [Configuring and Running the Load Generator Utility](#)
- [Creating a Load Generator Property File](#)
- [Creating a Data Feed File](#)
- [Configuring the csvgen Adapter in Your Application](#)

### Overview of Testing Applications With the Load Generator and csvgen Adapter

The load generator is a simple utility provided by Oracle Event Processing 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 Event Processing-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.

For more information on testing and debugging, see [Section , "Debugging an Oracle Event Processing Application Running on an Oracle Event Processing Server"](#).

### 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 Event Processing provides a default property file you can use if the default property values are adequate.

See [Section , "Creating a Load Generator Property File."](#)

2. Create a file that contains the actual data feed values.

See [Section , "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 Event Processing application.

See [Section , "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 , "Controlling Event Type Instantiation with an Event Type Builder Class"](#) for details.

5. Open a command window and set your environment as described in [Section , "Setting Your Development Environment"](#).

6. Change to the `ORACLE_CEP_HOME\ocep_11.1\utils\load-generator` directory, where `ORACLE_CEP_HOME` refers to the main Oracle Event Processing 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
```

## 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 Event Processing 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 Event Processing 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 21–1 Load Generator Properties**

Property	Description	Data Type	Required?
<code>test.csvDataFile</code>	Specifies the file that contains the data feed values.	String	Yes
<code>test.port</code>	The port number to which the load generator should send the data feed.  Each input adapter must be associated with its own <code>test.port</code> as <a href="#">Section , "Configuring the csvgen Adapter in Your Application"</a> describes.	Integer	Yes
<code>test.secs</code>	Total duration of the load generator run, in seconds. The default value is 30.	Integer	No
<code>test.rate</code>	Final data rate, in messages per second. The default value is 1.	Integer	No
<code>test.startRate</code>	Initial data rate, in messages per second. The default value is 1.	Integer	No
<code>test.rampUpSecs</code>	Number of seconds to ramp up from <code>test.startRate</code> to <code>test.rate</code> . The default value is 0.	Integer	No

## Creating a Data Feed File

A load generator data feed file contains the sample data feed values that correspond to the event type registered for your Oracle Event Processing application.

[Example 21–1](#) shows an `EmployeeEvent` and [Example 21–2](#) shows a load generator data feed file corresponding to this event type.

### Example 21–1 EmployeeEvent Event Type

```
<wlevs:event-type-repository>
  <wlevs:event-type type-name="EmployeeEvent">
    <wlevs:properties>
      <wlevs:property name="name" type="char" />
      <wlevs:property name="age" type="int" />
      <wlevs:property name="birthplace" type="char" length="512" />
    </wlevs:properties>
  </wlevs:event-type>
  ...
</wlevs:event-type-repository>
```

### Example 21–2 Data Feed File for EmployeeEvent Event Type

```
Lucy, 23, Madagascar
Nick, 44, Canada
Amanda, 12, Malaysia
```

Juliet, 43, Spain  
Horatio, 80, Argentina

A load generator data feed 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 commas as part of a string field.
- 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.
- By default, the maximum length of a string field is 256 characters.

To specify a longer string, set the `length` attribute of the `char` property in your event-type as [Example 21-1](#) shows for the `birthplace` property.

---

---

**Note:** The load generator does not fully comply with the CSV specification (<http://www.creativyst.com/Doc/Articles/CSV/CSV01.htm>).

---

---

For more information, see [Section , "Constraints on Event Types for Use With the csvgen Adapter"](#).

## 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 Event Processing-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 , "Creating a Load Generator Property File."](#)
- `wlevs:instance-property` element with name attribute `eventName` 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:metaData` element shown below:

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

```
<wlevs:property name="name" type="char"/>
<<wlevs:property name="age" type="int"/>
<<wlevs:property name="birthplace" type="char"/>
</wlevs:properties>
</wlevs:event-type>
</wlevs:event-type-repository>
```

This event type corresponds to the data feed file shown in [Section , "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="eventName" value="PersonType" />
  <wlevs:instance-property name="eventPropertyNames" value="name,age,birthplace" />
</wlevs:adapter>
```

Note how the bold 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 [Chapter 9, "Defining and Using Event Types"](#).



---

---

## Testing Applications With the Event Inspector

This chapter describes how to use the Event Inspector service to trace and inject events with any stage in an Oracle Event Processing event processing network (EPN).

This chapter includes the following sections:

- [Overview of Testing Applications With the Event Inspector](#)
- [Configuring the Event Inspector HTTP Pub-Sub Server](#)
- [Injecting Events](#)
- [Tracing Events](#)

---

---

**Note:** The Event Inspector service is not for use on a production Oracle Event Processing server. It is for use only during development.

---

---

### Overview of Testing Applications With the Event Inspector

Using the Event Inspector service, you can:

- View the events flowing out of any stage in the EPN
- Inject events into any stage in the EPN

You can use the Event Inspector service to test and debug Oracle CQL queries during development.

This section describes:

- [Section , "Tracing Events"](#)
- [Section , "Injecting Events"](#)
- [Section , "Event Inspector Event Types"](#)
- [Section , "Event Inspector HTTP Publish-Subscribe Channel and Server"](#)
- [Section , "Event Inspector Clients"](#)

For more information on testing and debugging, see [Section , "Debugging an Oracle Event Processing Application Running on an Oracle Event Processing Server"](#).

### Tracing Events

Using the Event Inspector service, you can view the events leaving any stage of the EPN.

The Event Inspector service uses a common HTTP pub-sub channel and server to trace events.

A trace event must have its `binding` attribute set to `outbound`.

For more information, see:

- [Section , "Event Inspector Event Types"](#)
- [Section , "Event Inspector HTTP Publish-Subscribe Channel and Server"](#)
- [Section , "Tracing Events"](#)

## Injecting Events

Using the Event Inspector service, you can inject events into any stage of the EPN.

The Event Inspector service uses a HTTP pub-sub channel and server to inject events.

An injected event must have its `binding` attribute set to `inbound`.

Using an Event Inspector client, you can inject:

- A single, simple event by type, such as the `StockTick` event.  
In this case, the specific event property types that you can use depends on the client.
- A single event directly to the HTTP pub-sub channel as a JSON-formatted character string.  
In this case, you can use any event property that JSON can represent.
- Multiple events using a file that contains one or more JSON-formatted character strings.  
In this case, you can use any event property that JSON can represent. The Event Inspector service client will parse the file and inject all its JSON strings to the HTTP pub-sub channel.  
You can use the GSON Java library to help you convert Java objects to JSON format when creating your input file.

For more information, see:

- <http://www.json.org/>
- <http://code.google.com/p/google-gson>
- [Section , "Event Inspector Event Types"](#)
- [Section , "Event Inspector HTTP Publish-Subscribe Channel and Server"](#)
- [Section , "Injecting Events"](#)

## Event Inspector Event Types

All Oracle Event Processing event types are supported: `JavaBean`, `Map`, and `tuple`.

The Event Inspector service converts events to the JavaScript Object Notation (JSON) format before publishing to the trace channel and you must inject events in JSON format.

JSON-formatted events must conform to the structure that [Example 22-1](#) shows. [Table 22-1](#) lists the required attributes.

---

---

**Note:** Byte arrays are not supported as property types in event types used with the event inspector.

---

---



**Example 22–1 Event Inspector JSON Event**

```

{
  "event-type": "myEventType",
  "operation": "insert",
  "binding": "outbound",
  "value":{
    "firstname": "Jane",
    "lastname": "Doe",
    "phone": {
      "code": 12345,
      "number": "office"
    },
  },
}

```

**Table 22–1 Event Inspector JSON Event Required Attributes**

Attribute	Description
event-type	The name of the Oracle Event Processing event as you defined it in the application assembly file's event-type-repository.
operation	Specify the type of event: <ul style="list-style-type: none"> <li>▪ insert: insert event.</li> <li>▪ delete: delete event</li> <li>▪ update: update event</li> <li>▪ heartbeat: heartbeat event</li> </ul>
binding	One of: <ul style="list-style-type: none"> <li>▪ inbound: injected event.</li> <li>▪ outbound: trace event.</li> </ul>
value	One or more JSON-formatted event properties as defined by the event-type.

For more information, see:

- <http://www.json.org/>
- [Section , "Overview of Oracle Event Processing Event Types"](#)
- [Section , "Tracing Events"](#)
- [Section , "Injecting Events"](#)

**Event Inspector HTTP Publish-Subscribe Channel and Server**

The Event Inspector service uses a dynamic HTTP publish-subscribe (HTTP pub-sub) channel (not configured in `config.xml`) that is named:

`/SERVERNAME/APPLICATIONNAME/STAGENAME/DIRECTION`

Where:

- **SERVERNAME**: the name of the Oracle Event Processing server on which the Oracle Event Processing EPN stage is running.
- **APPLICATIONNAME**: the name of the Oracle Event Processing application.
- **STAGENAME**: the name of the EPN stage.
- **DIRECTION**: one of either:
  - input: for event injection.

- output: for event tracing.

For example:

```
/server-1/myapp/MyInputAdapter/input
```

The Event Inspector service uses an HTTP pub-sub server. This can be any of:

- **Local:** you configure your `config.xml` file with an `event-inspector` element and configure its `pubsub-server-name` child element with the name of local pubsub server running on this machine. For more information, see [Section , "How to Configure a Local Event Inspector HTTP Pub-Sub Server"](#).
- **Remote:** you configure your `config.xml` file with an `event-inspector` element and configure its `pubsub-server-url` child element with a URL to an HTTP pub-sub server running on a remote machine. For more information, see [Section , "How to Configure a Remote Event Inspector HTTP Pub-Sub Server"](#).
- **Default:** if there is only one HTTP pub-sub server defined in your `config.xml` file and you do not specify a local or remote HTTP pub-sub server, then the Event Inspector service uses the local HTTP pub-sub server by default.

The Event Inspector service uses the same HTTP pub-sub channel and server for both tracing and injecting events.

For more information, see:

- [Section , "Tracing Events"](#)
- [Section , "Injecting Events"](#)

## Event Inspector Clients

The Event Inspector service supports the following clients:

- [Section , "Oracle Event Processing Visualizer"](#)

### Oracle Event Processing Visualizer

You can access the Event Inspector service using the Oracle Event Processing Visualizer.

For more information, see "Testing Applications With the Event Inspector" in the *Oracle Fusion Middleware Visualizer User's Guide for Oracle Event Processing*.

## Configuring the Event Inspector HTTP Pub-Sub Server

You can configure the Event Inspector service with a local or remote HTTP pub-sub server:

- [Section , "How to Configure a Local Event Inspector HTTP Pub-Sub Server"](#)
- [Section , "How to Configure a Remote Event Inspector HTTP Pub-Sub Server"](#)

You configure the Event Inspector HTTP pub-sub server in a component configuration file. For general information about these configuration files, see [Section , "Overview of Component Configuration Files."](#)

If there is only one HTTP pub-sub server defined in your `config.xml` and you do not specify a local or remote HTTP pub-sub server, then the Event Inspector service uses the local HTTP pub-sub server by default. For more information, see [Chapter 12, "Integrating an HTTP Publish-Subscribe Server"](#).

## How to Configure a Local Event Inspector HTTP Pub-Sub Server

You configure the Event Inspector service with a local HTTP pub-sub server in a component configuration file. Alternatively, you can configure a remote HTTP pub-sub server as [Section , "How to Configure a Remote Event Inspector HTTP Pub-Sub Server"](#) describes.

### To configure a local Event Inspector HTTP pub-sub server:

1. Open the EPN editor in the Oracle Event Processing IDE for Eclipse.  
See [Section , "Opening the EPN Editor"](#).
2. Right-click any component with a configuration file associated with it and select **Go to Configuration Source**.
3. Add an event-inspector element as [Example 22-2](#) shows.

#### **Example 22-2 Event Inspector Service Local HTTP Pub-Sub Server**

```
<event-inspector>
  <name>myEventInspectorConfig</name>
  <pubsub-server-name>myPubSub</pubsub-server-name>
</event-inspector>
```

Where the pubsub-server-name value myPubSub is the value of the http-pubsub element name child element as defined in the local Oracle Event Processing server config.xml file as [Example 22-3](#) shows.

#### **Example 22-3 Oracle Event Processing Built-In HTTP Pub-Sub Server http-pubsub Element**

```
...
<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>
  </pub-sub-bean>
</http-pubsub>
...
```

4. Save and close the config.xml file.

## How to Configure a Remote Event Inspector HTTP Pub-Sub Server

You configure the Event Inspector service with a remote HTTP pub-sub server in a component configuration file. Alternatively, you can configure a local HTTP pub-sub server as [Section , "How to Configure a Local Event Inspector HTTP Pub-Sub Server"](#) describes.

**To configure a Remote Event Inspector HTTP pub-sub server:**

1. Open the EPN editor in the Oracle Event Processing IDE for Eclipse.  
See [Section , "Opening the EPN Editor"](#).
2. Right-click any component with a configuration file associated with it and select **Go to Configuration Source**.
3. Add an event-inspector element as [Example 22–4](#) shows.

**Example 22–4 Event Inspector Service Remote HTTP Pub-Sub Server**

```
<event-inspector>
  <name>myEventInspectorTraceConfig</name>
  <pubsub-server-url>http://HOST:PORT/PATH</pubsub-server-url>
</event-inspector>
```

Where:

- *HOST*: is the host name or IP address of the remote Oracle Event Processing server.
- *PORT*: the remote Oracle Event Processing server `netio` port as defined in the remote Oracle Event Processing server `config.xml` file. Default: 9002.
- *PATH*: the value of the `http-pubsub` element `path` child element as defined in the remote Oracle Event Processing server `config.xml` file.

Given the `http-pubsub` configuration that [Example 22–3](#) shows, a valid `pubsub-server-url` would be:

```
http://remotehost:9002/pubsub
```

**Example 22–5 Oracle Event Processing Built-In HTTP Pub-Sub Server `http-pubsub` Element**

```
...
<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>
      ...
    </channels>
  </pub-sub-bean>
</http-pubsub>
...
```

4. Save and close the `config.xml` file.

## Injecting Events

After you configure the Event Inspector service HTTP pub-sub server, you can use Event Inspector clients to inject events. To configure event injection, you can use the Oracle Event Processing Visualizer or you can edit a component configuration file in

your application to specify injection settings that are in place when the application is deployed or redeployed.

Configure event injection in Oracle Event Processing Visualizer with settings that can be discarded when the application is redeployed. For more information on using Oracle Event Processing Visualizer to inject events, see "How to Inject a Simple Event on an Event Inspector Service Dynamic Channel" and "How to Inject an Event as a JSON String on an Event Inspector Service Dynamic Channel" in the *Oracle Fusion Middleware Visualizer User's Guide for Oracle Event Processing*.

If you want to specify event injection configuration settings that are in place when the application is deployed or redeployed, configure injection by editing component configuration settings for the stage to which you want to inject.

For example, the component configuration excerpt shown in [Section 22–6, "Event Injection Component Configuration Settings"](#) illustrates how you might configure a processor for event injection. The `inject-parameters` element's `active` child element specifies that injection is on, while the `channel-name` element specifies the HTTP pub-sub channel from which injected elements should be sent.

#### **Example 22–6 Event Injection Component Configuration Settings**

```
<processor>
  <name>FindCrossRates</name>
  <inject-parameters>
    <active>true</active>
    <channel-name>/NonClusteredServer/fx/FindCrossRates/output</channel-name>
  </inject-parameters>
  <rules>
    <!-- Query rules omitted. -->
  </rules>
</processor>
```

For reference information about these elements, see [Section , "inject-parameters"](#).

For more information, see:

- [Section , "Configuring the Event Inspector HTTP Pub-Sub Server"](#)

## Tracing Events

After you configure the Event Inspector service HTTP pub-sub server, you can use Event Inspector clients to trace events flowing out of any stage of your EPN. To trace events, you can either use the Oracle Event Processing Visualizer to configure tracing or you can edit a component configuration file in your application to specify trace settings that are in place when the application is deployed or redeployed.

Configure event tracing in Oracle Event Processing Visualizer with settings that can be discarded when the application is redeployed. For more information on using Oracle Event Processing Visualizer to trace events, see "How to Trace Events on a Dynamic Channel" in the *Oracle Fusion Middleware Visualizer User's Guide for Oracle Event Processing*.

For event tracing configuration that is in place when the application is deployed or redeployed, configure tracing by editing component configuration settings for the stage from which you want to trace.

For example, the component configuration excerpt shown in [Section 22–7, "Event Tracing Component Configuration Settings"](#) illustrates how you might configure a processor for event tracing. The `trace-parameters` element's `active` child element

specifies that tracing is on, while the channel-name element specifies the HTTP pub-sub channel to which traced elements should be sent.

**Example 22–7 Event Tracing Component Configuration Settings**

```
<processor>
  <name>FindCrossRates</name>
  <trace-parameters>
    <active>true</active>
    <channel-name>/NonClusteredServer/fx/FindCrossRates/output</channel-name>
  </trace-parameters>
  <rules>
    <!-- Query rules omitted. -->
  </rules>
</processor>
```

For reference information about these elements, see [Section , "trace-parameters"](#).

For more information, see:

- [Section , "Configuring the Event Inspector HTTP Pub-Sub Server"](#)

# Part IV

---

## Completing and Refining Oracle Event Processing Applications

Part IV contains the following chapters:

- [Chapter 23, "Assembling and Deploying Oracle Event Processing Applications"](#)
- [Chapter 24, "Developing Applications for High Availability"](#)
- [Chapter 25, "Developing Scalable Applications"](#)
- [Chapter 26, "Extending Component Configuration"](#)
- [Chapter 27, "Performance Tuning"](#)





---

---

# Assembling and Deploying Oracle Event Processing Applications

This chapter describes how to assemble and deploy Oracle Event Processing applications manually, by using the Oracle Event Processing IDE for Eclipse, and by using Oracle Event Processing Visualizer.

This chapter includes the following sections:

- [Overview of Application Assembly and Deployment](#)
- [Assembling an Oracle Event Processing Application](#)
- [Managing Application Libraries](#)
- [Managing Log Message Catalogs](#)
- [Deploying Oracle Event Processing Applications](#)

## 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 Event Processing. The term *application deployment* refers to the process of making an application available for processing client requests in an Oracle Event Processing domain.

This section describes:

- [Section , "Applications"](#)
- [Section , "Application Libraries"](#)
- [Section , "Application Dependencies"](#)
- [Section , "Deployment and Deployment Order"](#)
- [Section , "Configuration History Management"](#)

---

---

**Note:** Oracle Event Processing 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.

---

---

## Applications

In the context of Oracle Event Processing 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 Event Processing 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 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/wllevs` 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.

## Application Dependencies

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.

You define dependencies at design time.

This section describes:

- [Section , "Private Application Dependencies"](#)
- [Section , "Shared Application Dependencies"](#)
- [Section , "Native Code Dependencies"](#)

For more information, see:

- [Section , "Deployment and Deployment Order"](#)
- [Section , "Assembling an Oracle Event Processing Application"](#)

### Private Application Dependencies

Some dependencies are satisfied by a component bundled in and deployed with an application. For example, standard JAR files or property files.

For more information, see:

- [Section , "How to Add a Standard JAR File to an Oracle Event Processing Project"](#)
- [Section , "How to Add an OSGi Bundle to an Oracle Event Processing Project"](#)

- [Section , "How to Add a Property File to an Oracle Event Processing Project"](#)

### Shared Application Dependencies

Some dependencies are satisfied by a component deployed to the Oracle Event Processing server application library directory. These components are not bundled in and deployed with a specific application. Instead, they are accessible to any application that imports one or more of the packages that the application library exports.

For more information, see:

- [Section , "Native Code Dependencies"](#)
- [Section , "Application Libraries"](#)
- [Section , "How to Export a Package"](#)
- [Section , "How to Import a Package"](#)

### Native Code Dependencies

In some cases, you may create an application library that depends on native code libraries that you cannot or may not choose to package as application libraries.

In this case, you can put native code libraries in the operating system path (`bootclasspath`) of the Oracle Event Processing server when it is started, so that the native code libraries can be loaded by library bundles that need to call this native code.

For more information, see:

- [Section , "Shared Application Dependencies"](#)
- "Configuring the Oracle Event Processing Server Boot Classpath" in the *Oracle Fusion Middleware Administrator's Guide for Oracle Event Processing*

## Application Libraries

The Oracle Event Processing application library gives you a convenient location to deploy shared libraries and gives you complete control over the order in which shared libraries are deployed at Oracle Event Processing server start up time.

An application library is an OSGi bundle that contains a Java archive (JAR) of compiled Java classes and any other required artifacts.

You can use application libraries for a variety of purposes such as drivers or foreign stages (partial or complete Oracle Event Processing applications that are useful to other downstream applications).

Although you can add a library to a project as a simple embedded JAR file, there are advantages to using an application library, including:

- Simplifying application assembly and maintenance activities such as deploying an updated version of the library.
- Encouraging re-use.
- Reducing server disk space consumption.

You deploy application libraries to either of the following Oracle Event Processing server directories:

- [Section , "Library Directory"](#)
- [Section , "Library Extensions Directory"](#)

- [Section , "Creating Application Libraries"](#)

For more information, see:

- [Section , "Managing Application Libraries"](#)
- [Section , "Application Dependencies"](#)
- [Section , "Deployment and Deployment Order"](#)
- [Section , "Referencing Foreign Stages in an EPN Assembly File"](#)
- [Appendix A, "Additional Information about Spring and OSGi"](#)

### Library Directory

By default, the Oracle Event Processing server library directory is:

*DOMAIN\_DIR/servername/modules*

Where:

- *DOMAIN\_DIR*: is the domain directory such as `/oracle_cep/user_projects/domains/mydomain`.
- *servername*: is the server instance, such as `myserver`.

For example:

`/oracle_cep/user_projects/domains/mydomain/myserver/modules`

The libraries in this directory are deployed after the components in the library extensions directory but before any Oracle Event Processing applications.

If your library is a driver (such as a JDBC driver), you must put it in the library extensions directory as [Section , "Library Extensions Directory"](#) describes.

To configure the root of the application library directory path, see [Section , "How to Define the Application Library Directory Using Oracle Event Processing IDE for Eclipse"](#).

### Library Extensions Directory

By default, the Oracle Event Processing server library extensions directory is:

*DOMAIN\_DIR/servername/modules/ext*

Where:

- *DOMAIN\_DIR*: is the domain directory such as `/oracle_cep/user_projects/domains/mydomain`.
- *servername*: is the server instance, such as `myserver`.

For example:

`/oracle_cep/user_projects/domains/mydomain/myserver/modules/ext`

The libraries in this directory are deployed first along with the Oracle Event Processing server core modules.

If your library is a driver (such as a JDBC driver), you must put it in the library extensions directory so that it is activated in the correct order. For example, to override an older version with a newer version or to provide access to an alternative driver. For more information, see "Configuring Access to a Different Database Driver or Driver

Version" in the *Oracle Fusion Middleware Administrator's Guide for Oracle Event Processing*.

If your library is not a driver, you may put it in the library directory as [Section , "Library Directory"](#) describes.

To configure the root of the application library extensions directory path, see [Section , "How to Define the Application Library Directory Using Oracle Event Processing IDE for Eclipse"](#).

### Creating Application Libraries

Oracle Event Processing provides a `bundler.sh` utility you can use to create an OSGi bundle wrapper around an arbitrary Java Archive. The resultant bundle JAR may be deployed to an OSGi container where the Java packages/classes found within the bundle may be imported and utilized by other deployed bundles. An example use case is the packaging of third-party JDBC drivers.

The utility reads the specified source JAR file and creates a target JAR file that includes the content of the source JAR and a manifest with the appropriate bundle-related entries specified. All Java packages found in the source archive will be exported by the target bundle.

Optionally, a bundle activator can be generated that instantiates one or more classes found within the JAR and registers each object as an OSGi service. This feature provides the ability for component bundles to access and manipulate multiple versions of specific factory classes at runtime.

If you wish to manually configure the activator implementation, you can use the Oracle Event Processing IDE for Eclipse.

For more information, see:

- [Section , "How to Create an Application Library Using bundler.sh"](#)
- [Section , "How to Create an Application Library Using Oracle Event Processing IDE for Eclipse"](#)
- "How to Access a Database Driver Using an Application Library Built With `bundler.sh`" in the *Oracle Fusion Middleware Administrator's Guide for Oracle Event Processing*

## Deployment and Deployment Order

After you have assembled the application, you deploy it by making it known to the Oracle Event Processing domain using the deployment tool appropriate for your needs. For detailed instructions, see [Section , "Deploying Oracle Event Processing Applications."](#)

The Oracle Event Processing server deploys components in the following order at Oracle Event Processing server start up time:

1. Deploy libraries in the library extensions directory (`DOMAIN_DIR/servername/modules/ext` directory).
2. Deploy libraries in the library directory (`DOMAIN_DIR/servername/modules` directory).
3. Deploy Oracle Event Processing applications.

The Oracle Event Processing server deploys libraries from both the library extensions directory and library directory based on the lexical order of the library names. Lexical ordering includes the relative directory name plus JAR file name.

For example:

- `modules/a.jar` will start before `modules/b.jar`
- `modules/0/my.jar` will start before `module/my.jar` since `0/my.jar` comes before `my.jar` in lexical order

Using this convention, you can control the order in which Oracle Event Processing server deploys JAR files simply by organizing JAR files into appropriately named subdirectories of either the library extensions directory or library directory.

Once the application is deployed to Oracle Event Processing, the configured adapters immediately start listening for events for which they are configured, such as financial data feeds and so on.

For more information, see [Section , "Application Libraries"](#).

## Configuration History Management

When you deploy an application to the Oracle Event Processing server, the Oracle Event Processing server creates a configuration history for the application. Any configuration changes you make to rules or Oracle Event Processing high availability adapter configuration are recorded in this history. You can view and roll-back (undo) these changes using the Oracle Event Processing Visualizer or `wlevs.Admin` tool.

For more information, see:

- "Configuration History Management" in the *Oracle Fusion Middleware Visualizer User's Guide for Oracle Event Processing*
- "Commands for Managing Configuration History" in the *Oracle Fusion Middleware Administrator's Guide for Oracle Event Processing*

## Assembling an Oracle Event Processing Application

Assembling an Oracle Event Processing 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 Event Processing 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.

This section describes:

- [Section , "Assembling an Oracle Event Processing Application Using Oracle Event Processing IDE for Eclipse"](#)
- [Section , "Assembling an Oracle Event Processing Application Manually"](#)

- [Section , "Assembling Applications With Foreign Stages"](#)
- [Section , "Assembling a Custom Adapter or Event Bean in Its Own Bundle"](#)

## Assembling an Oracle Event Processing Application Using Oracle Event Processing IDE for Eclipse

You can use Oracle Event Processing IDE for Eclipse to easily assemble your Oracle Event Processing application.

For more information, see:

- [Section , "Exporting Oracle Event Processing Projects"](#)
- [Section , "Upgrading Projects"](#)
- [Section , "Managing Libraries and Other Non-Class Files in Oracle Event Processing Projects"](#)

If your application depends on foreign stages, see [Section , "Assembling Applications With Foreign Stages"](#).

## Assembling an Oracle Event Processing Application Manually

Optionally, you can assemble your Oracle Event Processing 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 Event Processing installation directory, such as `d:\oracle_cep`.

---



---

### To assemble an Oracle Event Processing application manually:

1. Open a command window and set your environment as described in [Section , "Setting Your Development Environment."](#)
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 , "Creating EPN Assembly Files"](#) 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.
8. Create a MANIFEST.MF file that contains descriptive information about the bundle.

See [Section , "Creating the MANIFEST.MF File."](#)

9. If you need to access third-party JAR files from your Oracle Event Processing application, see [Section , "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 Event Processing 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/Tasks/jar.html> for information on using the `jar` task or the Java SE documentation at

<http://download.oracle.com/javase/6/docs/technotes/tools/windows/jar.html> for information on using the `jar` command-line tool.

11. If your application depends on foreign stages, see [Section , "Assembling Applications With Foreign Stages"](#).

### 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 Event Processing.

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 Event Processing 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 Event Processing requires that you import the following packages at a minimum:

```
Import-Package:
com.bea.wlevs.adapter.defaultprovider;version="11.1.1",
com.bea.wlevs.ede;version="11.1.1",
com.bea.wlevs.ede.api;version="11.1.1",
com.bea.wlevs.ede.impl;version="11.1.1",
org.osgi.framework;version="1.3.0",
org.springframework.beans.factory;version="2.5.6",
org.apache.commons.logging;version="1.1.0",
com.bea.wlevs.spring;version="11.1.1",
com.bea.wlevs.util;version="11.1.1",
org.springframework.beans;version="2.5.6",
org.springframework.util;version="2.0",
org.springframework.core.annotation;version="2.5.6",
```



```
org.springframework.beans.factory;version="2.5.6",
org.springframework.beans.factory.config;version="2.5.6",
org.springframework.osgi.context;version="1.2.0",
org.springframework.osgi.service;version="1.2.0"
```

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="11.1.1",
com.bea.wlevs.configuration.application;version="11.1.1",
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: 1.6.0_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: 11.1.1.4_0
Bundle-ManifestVersion: 1
Bundle-Vendor: Oracle.
Bundle-Copyright: Copyright (c) 2006 by Oracle.
Import-Package: com.bea.wlevs.adapter.defaultprovider;version="11.1.1",
com.bea.wlevs.ede;version="11.1.1",
com.bea.wlevs.ede.impl;version="11.1.1",
com.bea.wlevs.ede.api;version="11.1.1",
org.osgi.framework;version="1.3.0",
org.apache.commons.logging;version="1.1.0",
com.bea.wlevs.spring;version="11.1.1",
com.bea.wlevs.util;version="11.1.1",
net.sf.cglib.proxy,
net.sf.cglib.core,
net.sf.cglib.reflect,
org.aopalliance.aop,
org.springframework.aop.framework;version="2.5.6",
org.springframework.aop;version="2.5.6",
org.springframework.beans;version="2.5.6",
org.springframework.util;version="2.0",
org.springframework.core.annotation;version="2.5.6",
org.springframework.beans.factory;version="2.5.6",
```

```
org.springframework.beans.factory.config;version="2.5.6",
org.springframework.osgi.context;version="1.2.0",
org.springframework.osgi.service;version="1.2.0",
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="11.1.1",
com.bea.wlevs.configuration.application;version="11.1.1",
com.sun.xml.bind.v2;version="2.0.2"
Bundle-Name: example.helloworld
Bundle-Description: WLEvS example helloworld
Bundle-SymbolicName: helloworld
```

### Accessing Third-Party JAR Files

When creating your Oracle Event Processing applications, you might need to access legacy libraries within existing third-party JAR files. You can ensure access to this legacy code using any of the following approaches:

- [Section , "Application Libraries"](#)
- [Section , "Accessing Third-Party JAR Files Using Bundle-Classpath"](#)
- [Section , "Accessing Third-Party JAR Files Using -Xbootclasspath"](#)

**Accessing Third-Party JAR Files Using Bundle-Classpath** The recommended approach is to package the third-party JAR files in your Oracle Event Processing application JAR file. You can put the JAR files anywhere you want.

---

---

**Note:** This approach gives you little control over the order in which JAR files are loaded and it is possible that dependency conflicts may occur. For this reason, Oracle recommends that you use the Oracle Event Processing server application library approach instead. For more information, see [Section , "Application Libraries"](#).

---

---

However, to ensure that your Oracle Event Processing 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-separated 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.

For more information, see [Section , "How to Add a Standard JAR File to an Oracle Event Processing Project"](#).

**Accessing Third-Party JAR Files Using -Xbootclasspath** If the JAR files include libraries used by *all* applications deployed to Oracle Event Processing, 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.

---

**Note:** This approach gives you little control over the order in which JAR files are loaded and it is possible that dependency conflicts may occur. For this reason, Oracle recommends that you use the Oracle Event Processing server application library approach instead. For more information, see [Section , "Application Libraries"](#).

---

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/ocp_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 Event Processing 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.

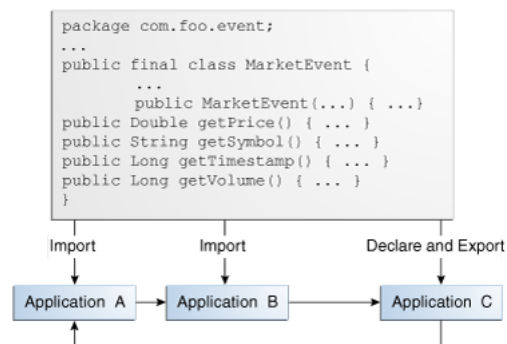
For example, if you want all deployed applications to be able to access a JAR file called `e:\jars\myExcitingJAR.jar`, update the `java` command in the start script as follows. The updated section is shown in bold (in practice, the command should be on one line):

```
%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
```

## Assembling Applications With Foreign Stages

When assembling applications that depend on foreign stages, be aware of classpath dependencies. Consider the application dependency graph that [Figure 23–1](#) shows.

**Figure 23–1 Foreign Stage Dependency Graph**



In this example, Application A depends on Application B, Application B depends on Application C, and Application C depends on Application A. Application C declares and exports an event type class for Java Bean event type `MarketEvent`. Applications A and B import the `MarketEvent` class that Application C provides.

Note the following:

- When you redeploy a foreign stage, you must redeploy all foreign stages that depend on that application or foreign stage.

For example, if you redeploy Application B, you must also redeploy Application A.

- If there is a classpath dependency between one foreign stage and another, when you deploy the foreign stage that declares and exports the shared class, you must redeploy all foreign stages that import the shared class.

For example, if you redeploy Application C, you must also redeploy Application A and B because Application A and B have a classpath dependency on Application C (`MarketEvent`).

For more information, see:

- [Section , "Referencing Foreign Stages in an EPN Assembly File"](#)
- [Section , "Deploying Oracle Event Processing Applications"](#)

## Assembling a Custom Adapter or Event Bean in Its Own Bundle

Typically, custom adapters and event beans are bundled in the same application JAR file that contains the other components of the EPN, 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.

This section describes:

- [Section , "How to Assemble a Custom Adapter in its Own Bundle"](#)
- [Section , "How to Assemble an Event Bean in its Own Bundle"](#)

### How to Assemble a Custom Adapter in its Own Bundle

You can assemble a custom adapter and its dependent classes in its own bundle.

#### To assemble a custom adapter in its own bundle:

1. Create an OSGI bundle that contains only the custom adapter Java class, the custom adapter factory Java class, and optionally, the event type Java class into which the custom adapter converts incoming data.

In this procedure, this bundle is called `GlobalAdapter`.

2. In the EPN assembly file of the `GlobalAdapter` bundle:
  - Register the adapter factory as an OSGI service as [Section , "Creating a Custom Adapter Factory"](#) describes.
  - If you are also including the event type in the bundle, register it as [Section , "Sharing Event Types Between Application Bundles"](#) describes.
  - Do *not* declare the custom adapter component using the `wlevs:adapter` element.

You will use this element in the EPN assembly file of the application bundle that actually uses the adapter.

- If you want to further configure the custom adapter, follow the usual procedure as [Section , "Configuring a Custom Adapter in a Component Configuration File"](#) describes.

- If you are including the event type in the `GlobalAdapter` bundle, export the `JavaBean` class in the `MANIFEST.MF` file of the `GlobalAdapter` bundle using the `Export-Package` header as [Section , "How to Export a Package"](#) describes.
- 3. Assemble and deploy the `GlobalAdapter` bundle as [Section , "Deploying Oracle Event Processing Applications"](#) describes.
- 4. In the EPN assembly file of the application that is going to use the custom adapter, declare the custom adapter component as [Section , "Configuring a Custom Adapter in an EPN Assembly File"](#) describes.

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.

- 5. 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 [Section , "Creating the MANIFEST.MF File"](#) describes.

### How to Assemble an Event Bean in its Own Bundle

You can assemble a custom event bean and its dependent classes in its own bundle.

#### To assemble a custom event bean in its own bundle:

- 1. Create an OSGI bundle that contains only the custom event bean Java class and the custom event bean factory Java class.

In this procedure, this bundle is called `GlobalEventBean`.

- 2. In the EPN assembly file of the `GlobalEventBean` bundle:
  - Register the custom event bean factory as an OSGI service as [Section , "Creating an Event Bean Factory"](#) describes.
  - Do *not* declare the custom event bean component using the `wlevs:event-bean` element.

You will use this element in the EPN assembly file of the application bundle that actually uses the event-bean.

- 3. Assemble and deploy the `GlobalEventBean` bundle as [Section , "Deploying Oracle Event Processing Applications"](#) describes.
- 4. In the EPN assembly file of the application that is going to use the custom event bean, declare the custom event bean component as [Section , "Creating an Event Bean Factory"](#) describes.

You still use the `provider` attribute to specify the OSGI-registered custom event bean factory, although in this case the OSGI registration happens in a different EPN assembly file (of the `GlobalEventBean` bundle) from the EPN assembly file that actually uses the adapter.

- 5. If you have exported the event type in the `GlobalEventBean` 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 [Section , "Creating the MANIFEST.MF File"](#) describes.

## Managing Application Libraries

The Oracle Event Processing application library gives you a convenient location to deploy shared libraries and gives you complete control over the order in which shared libraries are deployed at Oracle Event Processing server start up time.

This section describes how to manage an Oracle Event Processing server application library, including:

- [Section , "How to Define the Application Library Directory Using Oracle Event Processing IDE for Eclipse"](#)
- [Section , "How to Create an Application Library Using bundler.sh"](#)
- [Section , "How to Create an Application Library Using Oracle Event Processing IDE for Eclipse"](#)
- [Section , "How to Update an Application Library Using Oracle Event Processing IDE for Eclipse"](#)
- [Section , "How to View an Application Library Using the Oracle Event Processing Visualizer"](#)

For more information, see [Section , "Application Libraries"](#).

### How to Define the Application Library Directory Using Oracle Event Processing IDE for Eclipse

Before you can use the Oracle Event Processing server application library, you must update your Oracle Event Processing IDE for Eclipse design time configuration with the location of the application library directory.

For information on default application library configuration, see:

- [Section , "Library Directory"](#)
- [Section , "Library Extensions Directory"](#)

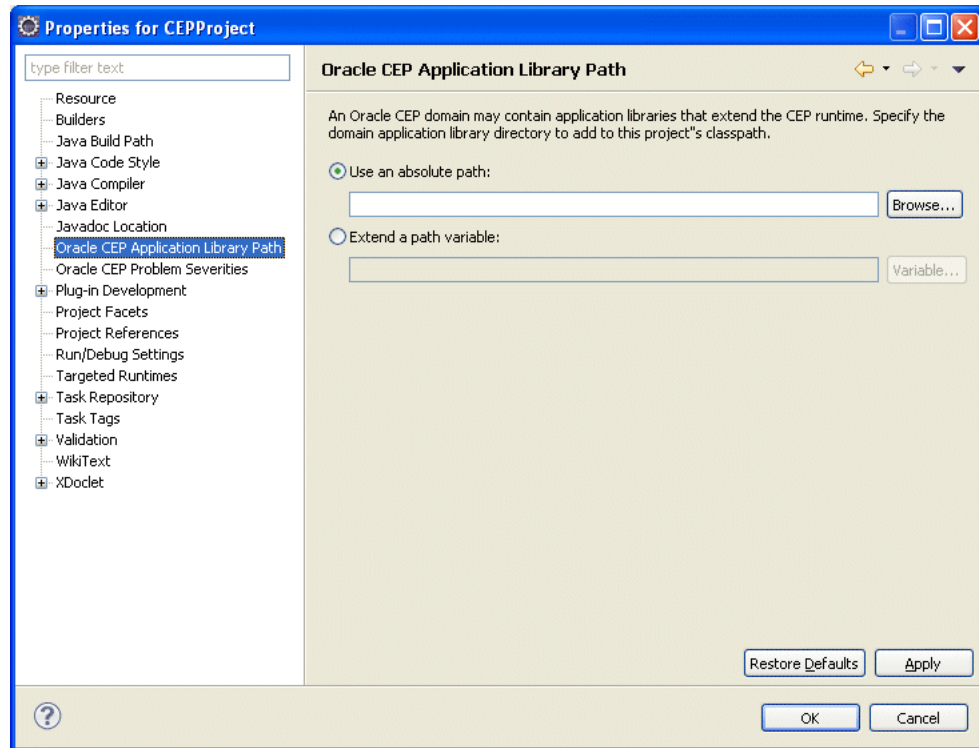
For more information, see [Section , "Managing Application Libraries"](#).

#### **To define an application library directory using Oracle Event Processing IDE for Eclipse:**

1. Launch the Oracle Event Processing IDE for Eclipse.
2. Right-click the project and select **Properties**.

The Preferences dialog appears as shown in [Figure 23–2](#).

**Figure 23–2 Preferences Dialog: Application Library Path**



3. Select **Oracle Oracle Event Processing Application Library Path**.
4. Specify the application library path as [Table 23–1](#) describes.

**Table 23–1 Oracle Event Processing Application Library Path**

Option	Description
Use an absolute path	Select this option to specify an absolute file path to the application library directory. See <a href="#">Section , "How to Configure an Absolute Path"</a> .
Extend a path variable	Select this option to specify an application library path based on a path variable. See <a href="#">Section , "How to Extend a Path Variable"</a> .

### How to Configure an Absolute Path

You can specify the application library path as an absolute file path. It may be more convenient in a team environment to specify the application library path based on a path variable as [Section , "How to Extend a Path Variable"](#) describes.

#### To configure an absolute path:

1. Click the **Browse** button to open a file system browser.
2. Use the file system browser to choose a directory.

---

**Note:** The directory must reside within an Oracle Event Processing server domain. For more information, see [Section , "Creating Oracle Event Processing Servers"](#).

---

3. Click **OK**.

4. Click **Apply**.
5. Click **OK**.

### How to Extend a Path Variable

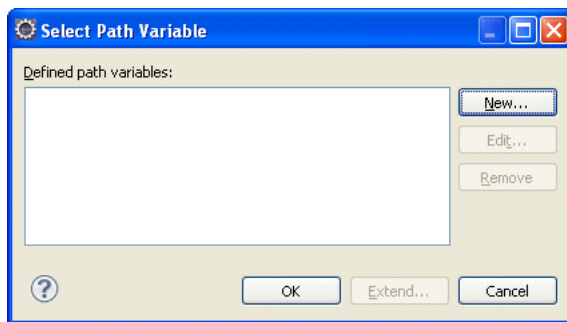
You can specify the application library path by extending a path variable. This is the most flexible approach and is appropriate for team environments. Alternatively, you can specify the application library with an absolute path as [Section , "How to Configure an Absolute Path"](#) describes.

#### To extend a path variable:

1. Click the **Variable** button.

The Select Path Variable dialog appears as [Figure 23–3](#) shows.

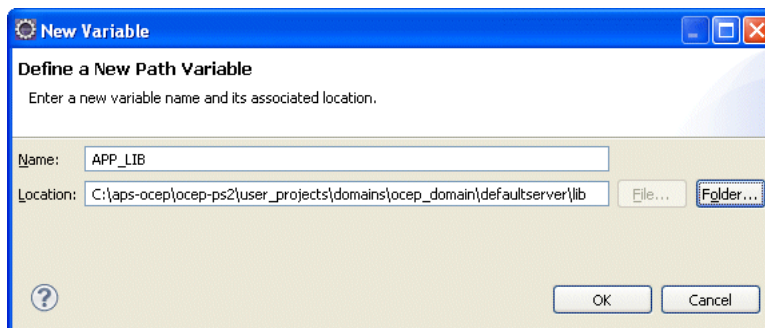
**Figure 23–3 Select Path Variable Dialog**



2. Click **New**.

The New Variable dialog appears as [Figure 23–4](#) shows.

**Figure 23–4 New Variable Dialog**



3. Configure the New Variable dialog as [Table 23–2](#) describes.

**Table 23–2 Oracle Event Processing Application Library Path Variable**

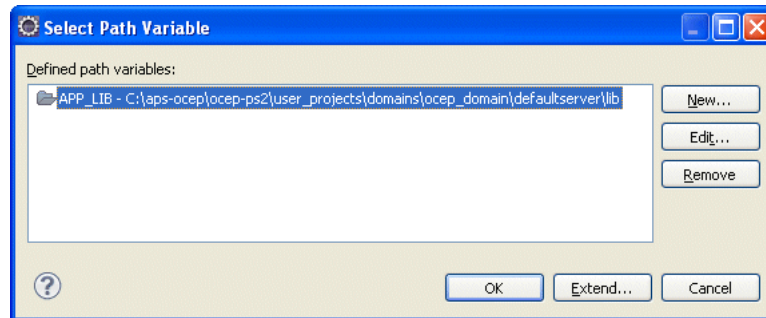
Option	Description
Name	Enter a name for the variable.
Location	Click the <b>F</b> older button to open a file system browser and choose the root directory to use as the application library directory.  <b>NOTE:</b> The directory must reside within an Oracle Event Processing server domain. For more information, see <a href="#">Section , "Creating Oracle Event Processing Servers"</a>



4. Click **OK**.

The new variable appears in the Select Path Variable dialog as [Figure 23–5](#) shows.

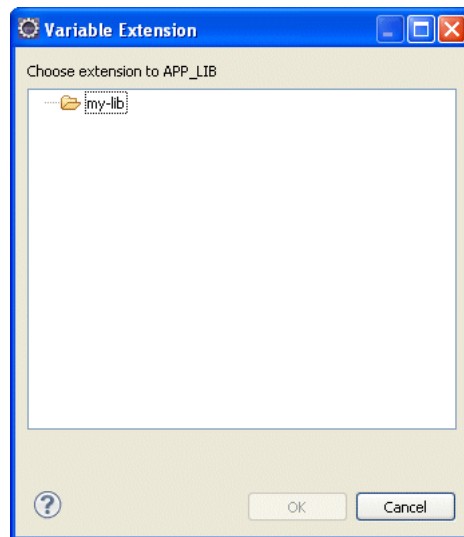
**Figure 23–5 Select Path Variable: With Variable**



5. Optionally, select the variable and click **Extend**.

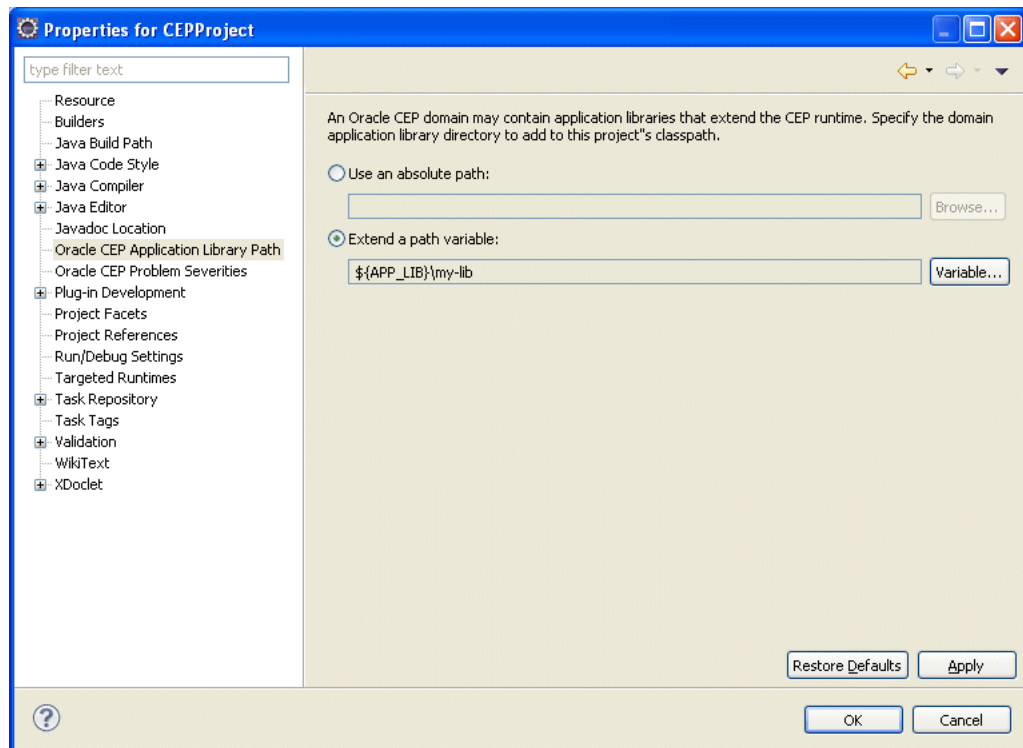
The Variable Extension dialog appears as [Figure 23–6](#) shows. This dialog shows any directories below the root directory you specified for this variable.

**Figure 23–6 Variable Extension Dialog**



6. Select a sub-directory and click **OK**.

The application library path is specified relative to the path variable you defined as [Figure 23–7](#) shows.

**Figure 23–7 Preferences Dialog: Application Library Path With Path Variable**

7. Click **Apply**.
8. Click **OK**.

## How to Create an Application Library Using `bundler.sh`

This procedure describes how to create an OSGi bundle using the `bundler` utility.

This is the preferred method. If you wish to manually configure the activator implementation, see [Section , "How to Create an Application Library Using Oracle Event Processing IDE for Eclipse"](#).

If you are creating an application library for a new JDBC driver, see "How to Access a Database Driver Using an Application Library Built With `bundler.sh`" in the *Oracle Fusion Middleware Administrator's Guide for Oracle Event Processing*.

For more information, see [Section , "Creating Application Libraries"](#).

### To create an application library using `bundler.sh`:

1. Set up your environment as described in [Section , "Setting Your Development Environment."](#)
2. Execute the `bundler.sh` script to create an OSGi bundle containing your driver.

The `bundler.sh` script is located in the `ORACLE_CEP_HOME/ocp_11.1/bin` directory, where `ORACLE_CEP_HOME` is the directory in which you installed the Oracle Event Processing server.

[Example 23–1](#) lists the `bundler.sh` command line options and [Table 23–3](#) describes them.

**Example 23–1 *bundler.sh* Command Line Options**

```

bundler -source JAR -name NAME -version VERSION
[-factory CLASS+] [-service INTERFACE+] [-fragmenthost HOST]
[-stagedir PATH] [-targetdir PATH]
[+import PACKAGE|REGEX+] [-imods REGEX;MODS+] [-import PACKAGE+]
[+export PACKAGE|REGEX+] [-emods REGEX;MODS+]
[-dimport PACKAGE+] [-explode] [-verbose]
    
```

**Table 23–3 *bundler.sh* Command Line Options**

Argument	Description
-source <i>JAR</i>	The path of the source JAR file to be bundled.
-name <i>NAME</i>	The symbolic name of the bundle. The root of the target JAR file name is derived from the name value.
-version <i>VERSION</i>	The bundle version number. All exported packages are qualified with a version attribute with this value. The target JAR file name contains the version number.
-factory <i>CLASS+</i>	An optional argument that specifies a space-delimited list of one or more factory classes that are to be instantiated and registered as OSGi services. Each service is registered with the OSGi service registry with name (-name) and version (-version) properties. This argument is incompatible with the -fragmenthost argument.
-service <i>INTERFACE+</i>	An optional argument that specifies a space-delimited list of one or more Java interfaces that are used as the object class of each factory object service registration. If no interface names are specified, or the number of interfaces specified does not match the number of factory classes, then each factory object will be registered under the factory class name.
-fragmenthost <i>HOST</i>	An optional argument indicating that the resultant bundle is a fragment bundle and specifies the symbolic name of the host bundle. This argument is incompatible with the -factory argument.
-stagedir <i>PATH</i>	An optional argument that specifies where to write temporary files when creating the target JAR file. Default: ./bundler.tmp
-targetdir <i>PATH</i>	An optional argument that specifies the location of the generated bundle JAR file. Default: current working directory (.).
+import <i>PACKAGE REGEX+</i>	A space-delimited list of one or more packages or regular expressions that select the packages to <i>exclude</i> from the manifest Import-Package attribute. By default, all dependent packages will be imported (except java.*).
-imods <i>REGEX;MODS+</i>	The import modifiers will be applied to the packages matching regular expression.
-import <i>PACKAGE</i>	Additional packages to include on the manifest Import-Package attribute. Note that any specified import modifiers will not be applied.
+export <i>PACKAGE REGEX+</i>	A space-delimited list of one or more packages or regular expressions that select the packages to <i>exclude</i> from the manifest Export-Package attribute. By default, all bundle packages will be exported.
-emods <i>REGEX;MODS+</i>	The export modifiers will be applied to the packages matching regular expression.
-dimport <i>PACKAGE+</i>	Packages to include on the manifest DynamicImport-Package attribute.
-explode	This optional flag specifies that the content of the source JAR should be exploded into the target JAR file. By default, the source JAR is nested within the target JAR file and the generated bundle manifest will contain an appropriate Bundle-Classpath attribute.
-verbose	An optional flag to enable verbose output.

[Example 23–2](#) shows how to use the `bundler.sh` to create an OSGi bundle for an Oracle JDBC driver.

**Example 23–2 Using the Bundler Utility**

```
bundler.sh \
-source C:\drivers\com.oracle.ojdbc14_11.2.0.jar \
-name oracle11g \
-version 11.2.0 \
-factory oracle.jdbc.xa.client.OracleXADataSource oracle.jdbc.OracleDriver \
-service javax.sql.XADataSource java.sql.Driver \
-targetdir C:\stage
```

The source JAR is an Oracle driver located in directory `C:\drivers`. The name of the generated bundle JAR is the concatenation of the `-name` and `-version` arguments (`oracle10g_11.2.0.jar`) and is created in the `C:\stage` directory. The bundle JAR contains the files that [Example 23–3](#) shows.

**Example 23–3 Bundle JAR Contents**

```
1465 Thu Jun 29 17:54:04 EDT 2006 META-INF/MANIFEST.MF
1540457 Thu May 11 00:37:46 EDT 2006 com.oracle.ojdbc14_11.2.0.jar
1700 Thu Jun 29 17:54:04 EDT 2006 com/bea/core/tools/bundler/Activator.class
```

The command line options specify that there are two factory classes that will be instantiated and registered as an OSGi service when the bundle is activated, each under a separate object class as [Table 23–4](#) shows.

**Table 23–4 Factory Class and Service Interfaces**

Factory Class	Service Interface
<code>oracle.jdbc.xa.client.OracleXADataSource</code>	<code>javax.sql.XADataSource</code>
<code>oracle.jdbc.OracleDriver</code>	<code>java.sql.Driver</code>

Each service registration will be made with a name property set to `oracle11g` and a version property with a value of `11.2.0`. [Example 23–4](#) shows the Oracle Event Processing server log messages showing the registration of the services.

**Example 23–4 Service Registration Log Messages**

```
...
INFO: [Jun 29, 2006 5:54:18 PM] Service REGISTERED: { version=11.2.0, name=oracle11g,
objectClass=[ javax.sql.XADataSource ], service.id=23 }
INFO: [Jun 29, 2006 5:54:18 PM] Service REGISTERED: { version=11.2.0, name=oracle11g,
objectClass=[ java.sql.Driver ], service.id=24 }
INFO: [Jun 29, 2006 5:54:18 PM] Bundle oracle11g STARTED
...
```

3. Copy the application library JAR to the appropriate Oracle Event Processing server application library directory:
  - a. If your bundle is a driver, you must put it in the library extensions directory. See [Section , "Library Extensions Directory"](#).
  - b. If your bundle is not a driver, you may put it in the library directory. See [Section , "Library Directory"](#)

For more information, see [Section , "Application Libraries"](#).
4. Stop and start the Oracle Event Processing server.

See "Starting and Stopping Oracle Event Processing Servers" in the *Oracle Fusion Middleware Administrator's Guide for Oracle Event Processing*.

## How to Create an Application Library Using Oracle Event Processing IDE for Eclipse

This procedure describes how to create an OSGi bundle for your driver using the Oracle Event Processing IDE for Eclipse and deploy it on the Oracle Event Processing server.

This is the preferred method. If do not wish to manually configure the activator implementation, see [Section , "How to Create an Application Library Using bundler.sh"](#).

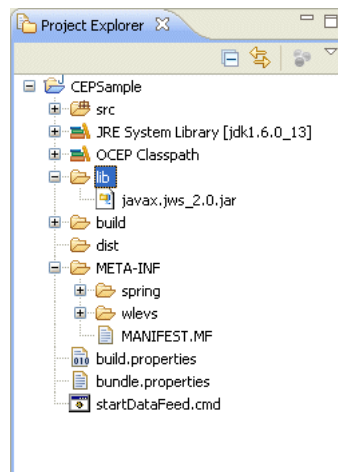
If you are creating an application library for a new JDBC driver, see "How to Access a Database Driver Using an Application Library Built With Oracle Event Processing IDE for Eclipse" in the *Oracle Fusion Middleware Administrator's Guide for Oracle Event Processing*.

### To create an application library using Oracle Event Processing IDE for Eclipse:

1. Using the Oracle Event Processing IDE for Eclipse, create a new Oracle Event Processing project.  
For more information, [Section , "Creating Oracle Event Processing Projects"](#).
2. Right-click your project folder and select **New > Folder**.
3. Enter `lib` in the **Folder name** field and click **Finish**.
4. Outside of the Oracle Event Processing IDE for Eclipse, copy your JDBC JAR file into the `lib` folder.
5. Inside the Oracle Event Processing IDE for Eclipse, right-click the `lib` folder and select **Refresh**.

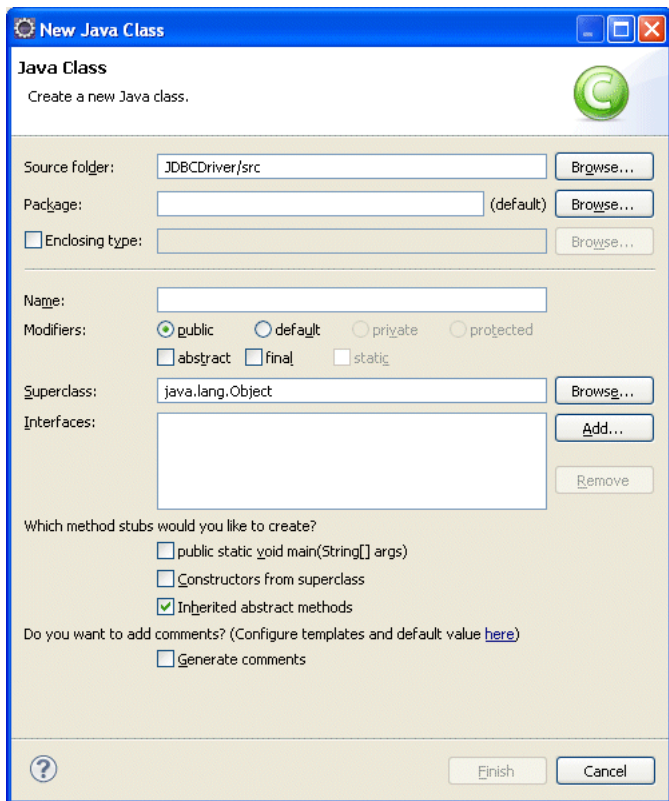
The JAR file appears in the `lib` folder as [Figure 23–8](#) shows.

**Figure 23–8 Oracle Event Processing IDE for Eclipse lib Directory**



6. Right-click the `src` directory and select **New > Class**.  
The Java Class dialog appears as [Figure 23–9](#) shows.

**Figure 23–9 New Java Class Dialog**



7. Configure the New Java Class dialog as Table 23–5 shows.

**Table 23–5 New Java Class Parameters**

Parameter	Description
Package	The package name. For example, com.foo.
Name	The name of the class. For example, MyActivator.

Leave the other parameters at their default values.

8. Click **Finish**.

A new Java class is added to your project.

9. Edit the Java class to implement it as Example 23–5 shows.

Be sure to set the NAME and VERSION so that they supersede the existing version of JDBC driver. In this example, the existing version is:

- oracle10g
- 10.0.0

To supersede the existing version, the MyActivator class sets these values to:

- oracle11g
- 11.2.0

**Example 23–5 MyActivator Class Implementation**

```
package com.foo;
```

```
import java.util.Dictionary;
import java.util.Properties;

import javax.sql.XADataSource;
import java.sql.Driver;

import org.osgi.framework.BundleActivator;
import org.osgi.framework.BundleContext;
import org.osgi.framework.ServiceRegistration;

public class MyActivator implements BundleActivator {

    private static final String NAME="oracle11g";
    private static final String VERSION="11.2.0";

    private String[] factories =
{"oracle.jdbc.xa.client.OracleXADataSource", "oracle.jdbc.OracleDriver"};
    private String[] interfaces= {"javax.sql.XADataSource", "java.sql.Driver"};
    private ServiceRegistration[] serviceRegistrations = new
ServiceRegistration[factories.length];

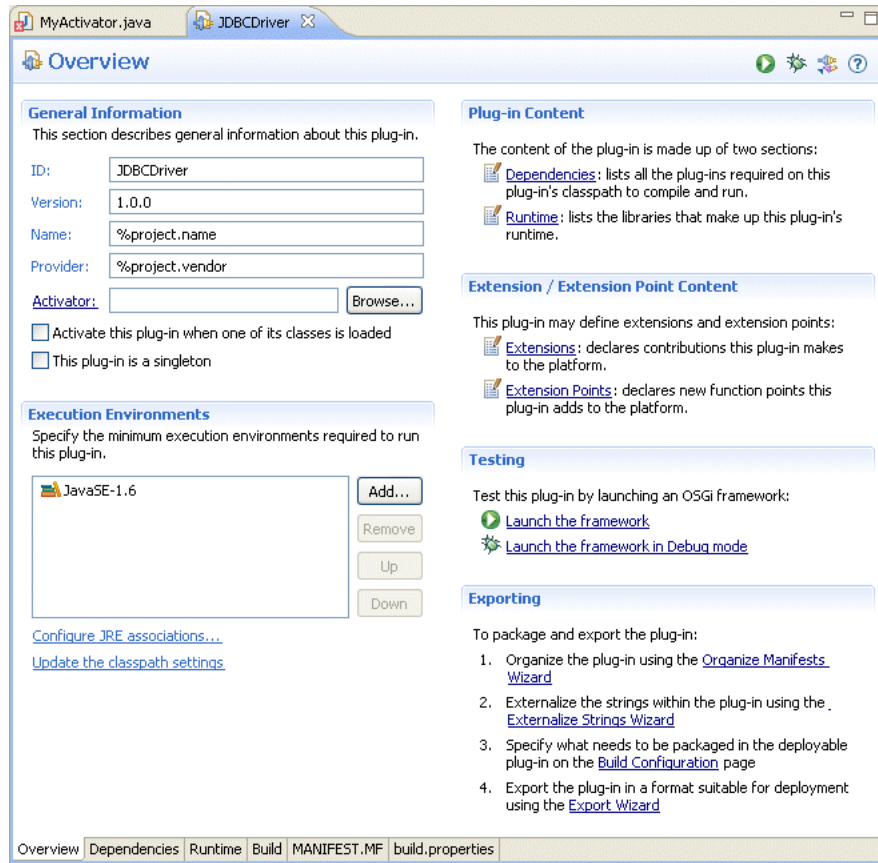
    public void start(BundleContext bc) throws Exception {
        Dictionary props = new Properties();
        props.put("name", NAME);
        props.put("version", VERSION);
        for (int i=0; i<factories.length; i++) {
            Object svc = bc.getBundle().loadClass(factories[i]).newInstance();
            serviceRegistrations[i] = bc.registerService(interfaces[i], svc, props);
        }
    }

    public void stop(BundleContext bc) throws Exception {
        for (int i=0; i<serviceRegistrations.length; i++) {
            serviceRegistrations[i].unregister();
        }
    }
}
```

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

The Manifest Editor appears as [Figure 23–10](#) shows.

**Figure 23–10 Manifest Editor: Overview Tab**

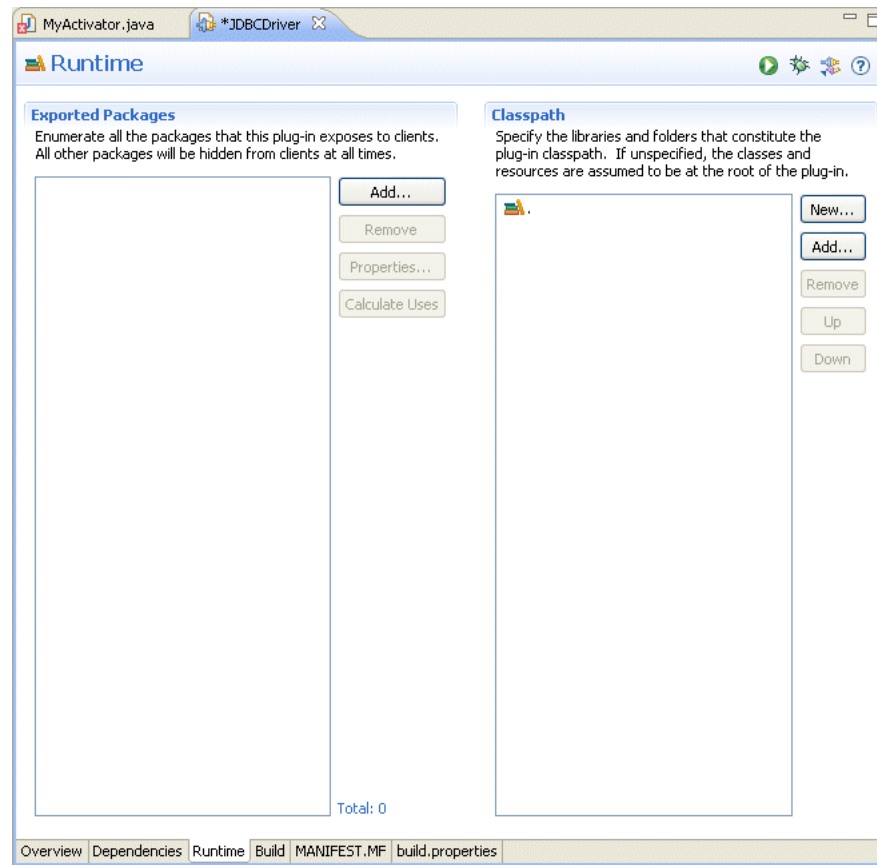


11. Click the **Runtime** tab.

The Runtime tab appears as [Figure 23–11](#) shows.



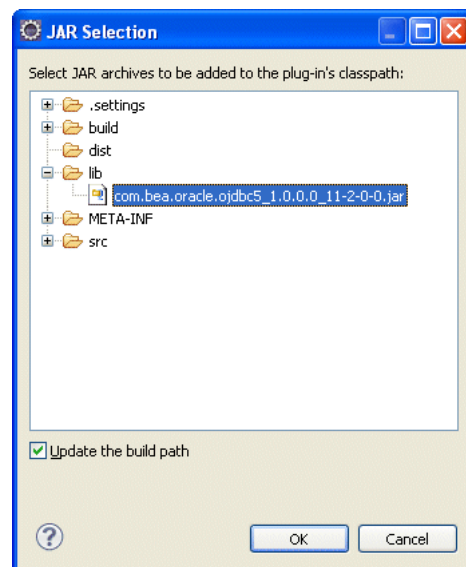
**Figure 23–11 Manifest Editor: Runtime Tab**



12. In the Classpath pane, click **Add**.

The JAR Selection dialog appears as [Figure 23–12](#) shows.

**Figure 23–12 JAR Selection Dialog**

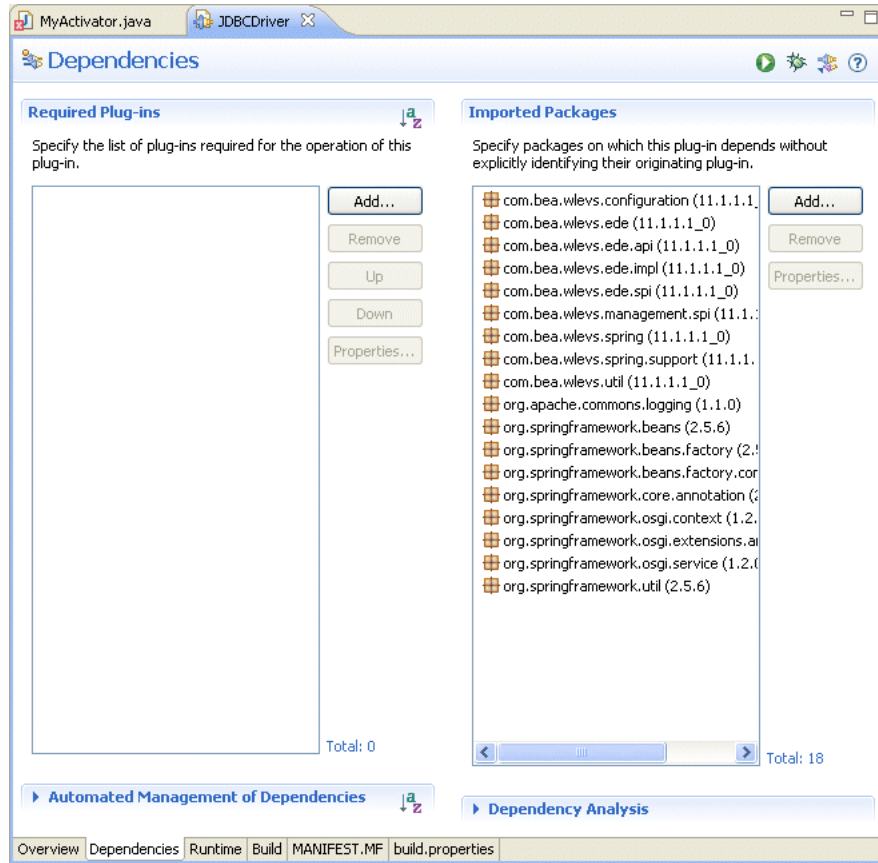


13. Expand the **lib** directory and select your database driver JAR file.

14. Click **OK**.
15. Click the **Dependencies** tab.

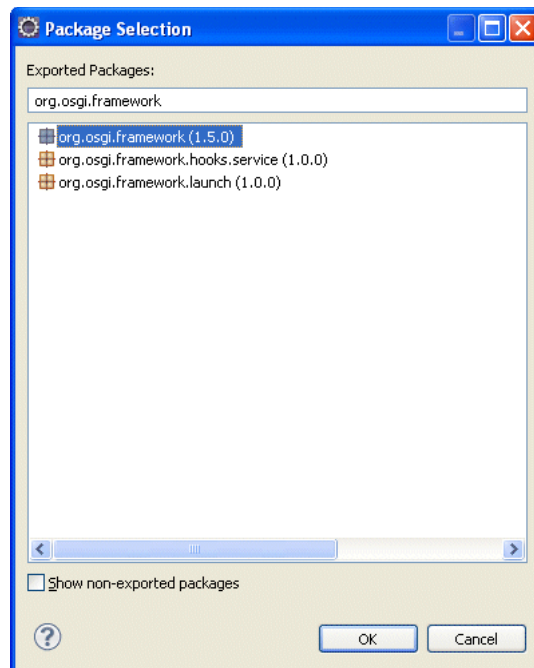
The Dependencies tab appears as [Figure 23–13](#) shows.

**Figure 23–13 Manifest Editor: Dependencies Tab**



16. In the Imported Packages pane, click **Add**.

The Package Selection dialog appears as [Figure 23–14](#) shows.

**Figure 23–14 Package Selection Dialog**

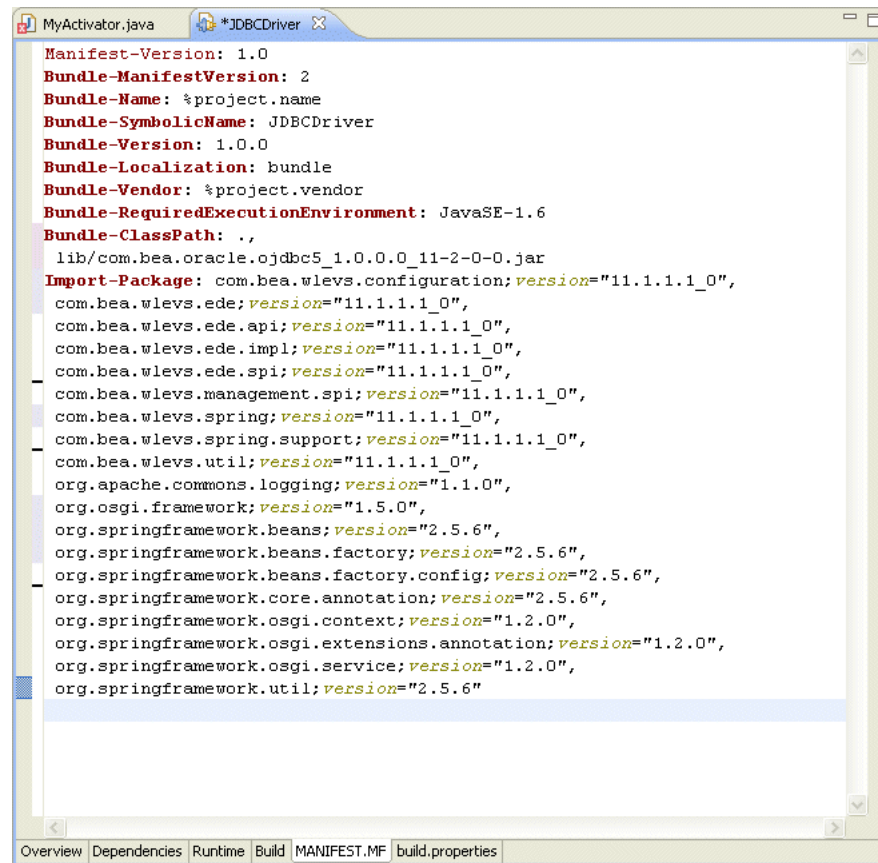
17. In the **Exported Packages** field, enter `org.osgi.framework`.

The list box shows all the packages with that prefix as [Figure 23–14](#) shows.

18. Select `org.osgi.framework` in the list box and click **OK**.

19. Click the **MANIFEST.MF** tab.

The `MANIFEST.MF` tab appears as [Figure 23–15](#) shows.

**Figure 23–15 Manifest Editor**

20. Un-JAR your JAR to a temporary directory as [Example 23–6](#) shows.

**Example 23–6 Un-JAR the Database Driver**

```

$ pwd
/tmp
$ ls com.*
com.bea.oracle.ojdbc6_1.0.0.0_11-1-0-7.jar
$ mkdir driver
$ cd driver
$ jar -xvf ../com.bea.oracle.ojdbc6_1.0.0.0_11-1-0-7.jar
$ ls
META-INF oracle
$ cd META-INF
$ ls
MANIFEST.MF services

```

21. Open your JAR MANIFEST.MF file and copy its **Export-Package** entry and paste it into the Manifest Editor as [Example 23–7](#) shows.

**Example 23–7 Adding Export-Package to the Manifest Editor**

```

Manifest-Version: 1.0
Bundle-ManifestVersion: 2
Bundle-Name: %project.name
Bundle-SymbolicName: JDBCDriver
Bundle-Version: 1.0.0
Bundle-Localization: bundle
Bundle-Vendor: %project.vendor

```

```

Bundle-RequiredExecutionEnvironment: JavaSE-1.6
Bundle-ClassPath: .
Import-Package: com.bea.wlevs.configuration;version="11.1.1.4_0", ...
Export-Package: oracle.core.lmx;version=1.0.0.0_11-1-0-7,oracle.core.l
vf;version=1.0.0.0_11-1-0-7,oracle.jdbc;version=1.0.0.0_11-1-0-7,orac
le.jdbc.aq;version=1.0.0.0_11-1-0-7,oracle.jdbc.connector;version=1.0
.0.0_11-1-0-7,oracle.jdbc.dcn;version=1.0.0.0_11-1-0-7,oracle.jdbc.dr
iver;version=1.0.0.0_11-1-0-7,oracle.jdbc.internal;version=1.0.0.0_11
-1-0-7,oracle.jdbc.oci;version=1.0.0.0_11-1-0-7,oracle.jdbc.oracore;v
ersion=1.0.0.0_11-1-0-7,oracle.jdbc.pool;version=1.0.0.0_11-1-0-7,ora
cle.jdbc.rowset;version=1.0.0.0_11-1-0-7,oracle.jdbc.util;version=1.0
.0.0_11-1-0-7,oracle.jdbc.xa;version=1.0.0.0_11-1-0-7,oracle.jdbc.xa.
client;version=1.0.0.0_11-1-0-7,oracle.jpub.runtime;version=1.0.0.0_1
1-1-0-7,oracle.net.ano;version=1.0.0.0_11-1-0-7,oracle.net.aso;versio
n=1.0.0.0_11-1-0-7,oracle.net.jndi;version=1.0.0.0_11-1-0-7,oracle.ne
t.ns;version=1.0.0.0_11-1-0-7,oracle.net.nt;version=1.0.0.0_11-1-0-7,
oracle.net.resolver;version=1.0.0.0_11-1-0-7,oracle.security.o3logon;
version=1.0.0.0_11-1-0-7,oracle.security.o5logon;version=1.0.0.0_11-1
-0-7,oracle.sql;version=1.0.0.0_11-1-0-7,oracle.sql.converter;version
=1.0.0.0_11-1-0-7

```

22. Add a Bundle-Activator element to the Manifest Editor as [Example 23–8](#) shows.

The value of the Bundle-Activator is the fully qualified class name of your Activator class.

#### **Example 23–8 Adding a Bundle-Activator Element to the Manifest Editor**

```

Manifest-Version: 1.0
Bundle-Activator: com.foo.MyActivator
Bundle-ManifestVersion: 2
Bundle-Name: %project.name
Bundle-SymbolicName: JDBCDBriver
Bundle-Version: 1.0.0
Bundle-Localization: bundle
Bundle-Vendor: %project.vendor
Bundle-RequiredExecutionEnvironment: JavaSE-1.6
Bundle-ClassPath: .
Import-Package: com.bea.wlevs.configuration;version="11.1.1.4_0", ...
Export-Package: oracle.core.lmx;version=1.0.0.0_11-1-0-7, ...
...

```

23. Add a DynamicImport-Package element to the Manifest Editor as [Example 23–9](#) shows.

#### **Example 23–9 Adding a DynamicImport-Package Element to the Manifest Editor**

```

Manifest-Version: 1.0
Bundle-Activator: com.foo.MyActivator
Bundle-ManifestVersion: 2
Bundle-Name: %project.name
Bundle-SymbolicName: JDBCDBriver
Bundle-Version: 1.0.0
Bundle-Localization: bundle
Bundle-Vendor: %project.vendor
Bundle-RequiredExecutionEnvironment: JavaSE-1.6
Bundle-ClassPath: .
DynamicImport-Package: *
Import-Package: com.bea.wlevs.configuration;version="11.1.1.4_0", ...
Export-Package: oracle.core.lmx;version=1.0.0.0_11-1-0-7, ...
...

```

24. Export your Oracle Event Processing application to a JAR file.

For more information, see [Section , "How to Export an Oracle Event Processing Project"](#).

25. Copy the bundler JAR to the appropriate Oracle Event Processing server application library directory:
  - a. If your bundle is a driver, you must put it in the library extensions directory. See [Section , "Library Extensions Directory"](#).
  - b. If your bundle is not a driver, you may put it in the library directory. See [Section , "Library Directory"](#)

For more information, see [Section , "Application Libraries"](#).

26. Stop and start the Oracle Event Processing server.

See "Starting and Stopping Oracle Event Processing Servers" in the *Oracle Fusion Middleware Administrator's Guide for Oracle Event Processing*.

## How to Update an Application Library Using Oracle Event Processing IDE for Eclipse

When you add, replace, or remove a JAR file in the application library extension or application library directory or their user-defined subdirectories, you must make this change in two places:

- On the local Oracle Event Processing server you used to create a server runtime in the Oracle Event Processing IDE for Eclipse.
- On the production Oracle Event Processing server to which you deploy dependent applications.

These changes need not be performed simultaneously: you must make the change to the local Oracle Event Processing server before making code changes to projects that depend on the application library change; you must make the change to the production Oracle Event Processing server before you deploy applications that depend on the application library change.

For more information, see [Section , "Managing Application Libraries"](#).

### To update an application library using Oracle Event Processing IDE for Eclipse:

1. Add a new or revised bundle to the application library extension or application library directory on the production Oracle Event Processing server.

This is the server to which you will deploy applications that depend on this application library.

To control library deployment order, organize your libraries in appropriately named subdirectories.

For more information, see:

- [Section , "Library Extensions Directory"](#)
- [Section , "Library Directory"](#)
- [Section , "Deployment and Deployment Order"](#)

2. Stop and start the production Oracle Event Processing server.

The Oracle Event Processing server refreshes itself from the updated application library extension or application library directory.

For more information, see:

- "Starting and Stopping an Oracle Event Processing Server in a Standalone-Server Domain" in the *Oracle Fusion Middleware Administrator's Guide for Oracle Event Processing*
  - "Starting and Stopping an Oracle Event Processing Server in a Multi-Server Domain" in the *Oracle Fusion Middleware Administrator's Guide for Oracle Event Processing*
3. Add the same new or revised bundle to the application library extension or application library directory on your Oracle Event Processing IDE for Eclipse targeted runtime Oracle Event Processing server.
  4. Start the Oracle Event Processing IDE for Eclipse.
  5. Right click a project and select **Refresh Targeted Runtimes**.  
The Oracle Event Processing IDE for Eclipse refreshes this project from the updated application library extension or application library directory on your Oracle Event Processing IDE for Eclipse targeted runtime Oracle Event Processing server.
  6. If necessary, update your application's dependencies.  
For example, if you added a new bundle or changed the version of an existing bundle.  
For more information, see [Section , "Application Dependencies"](#).
  7. Assemble and deploy your application to the production Oracle Event Processing server.  
For more information, see [Section , "Deploying Oracle Event Processing Applications"](#).  
The dependencies you defined for this application in the Oracle Event Processing IDE for Eclipse at development time will be satisfied by the components you installed in the application library of your production Oracle Event Processing server at runtime.

## How to View an Application Library Using the Oracle Event Processing Visualizer

Using the Oracle Event Processing Visualizer, you can view the application libraries deployed to the Oracle Event Processing server.

You can view libraries from both the library extensions directory and libraries directory.

---



---

**Note:** You cannot deploy an application library to an Oracle Event Processing server using the Oracle Event Processing Visualizer. You may only deploy Oracle Event Processing applications to an Oracle Event Processing server using the Oracle Event Processing Visualizer.

---



---

For more information, see:

- "How to View the Application Libraries Deployed to an Oracle Event Processing Server" in the *Oracle Fusion Middleware Visualizer User's Guide for Oracle Event Processing*
- [Section , "Library Extensions Directory"](#)
- [Section , "Library Directory"](#)

- [Section , "Managing Application Libraries"](#)

## Managing Log Message Catalogs

This section describes how to manage log message catalogs that you can use to localize an Oracle Event Processing application, including:

- [Section , "Using Message Catalogs With Oracle Event Processing Server"](#)
- [Section , "How to Parse a Message Catalog to Generate Logger and TextFormatter Classes for Localization"](#)

For more information, see:

- "Configuring Logging and Debugging for Oracle Event Processing" in the *Oracle Fusion Middleware Administrator's Guide for Oracle Event Processing*
- [Appendix G, "Schema Reference: Message Catalog msgcat.dtd"](#)
- [Appendix H, "Schema Reference: Locale Message Catalog l10n\\_msgcat.dtd"](#)

## Using Message Catalogs With Oracle Event Processing Server

A message catalog is a single XML file that contains a collection of text messages, with each message indexed by a unique identifier. You compile these XML files into classes using `weblogic.i18ngen` during the build process. (See `weblogic.i18ngen` Utility for more information). The methods of the resulting classes are the objects used to log messages at runtime.

Message catalogs support multiple locales or languages. For a specific message catalog there is exactly one default version, known as the top-level catalog, which contains the English version of the messages. Then there are corresponding locale-specific catalogs, one for each additional supported locale.

The top-level catalog (English version) includes all the information necessary to define the message. The locale-specific catalogs contain only the message ID, the date changed, and the translation of the message for the specific locale.

The message catalog files are defined by one of the following XML document type definition (DTD) files:

- `msgcat.dtd` - Describes the syntax of top-level, default catalogs.
- `l10n_msgcat.dtd` - Describes the syntax of locale-specific catalogs.

The DTDs are stored in `ORACLE_CEP_HOEM/modules/com.bea.core.i18n.generator_1.4.0.0.jar`.

You can create a single log message catalog for all logging requirements, or create smaller catalogs based on a subsystem or Java package. Oracle recommends using multiple subsystem catalogs so you can focus on specific portions of the log during viewing.

For simple text catalogs, we recommend creating a single catalog for each utility being internationalized

This section describes:

- [Section , "Message Catalog Hierarchy"](#)
- [Section , "Guidelines for Naming Message Catalogs"](#)
- [Section , "Using Message Arguments"](#)



- [Section , "Message Catalog Formats"](#)
- [Section , "Message Catalog Localization"](#)

For more information, see:

- [Appendix G, "Schema Reference: Message Catalog msgcat.dtd"](#)
- [Appendix H, "Schema Reference: Locale Message Catalog l10n\\_msgcat.dtd"](#)

## Message Catalog Hierarchy

All messages must be defined in the default, top-level catalog.

Catalogs that provide different localizations of the base catalogs are defined in msgcat subdirectories named for the locale (for example, msgcat/de for Germany). You might have a top-level catalog named mycat.xml, and a German translation of it called .de/mycat.xml. Typically the top-level catalog is English. However, English is not required for any catalogs.

Locale designations (for example, de) also have a hierarchy as defined in the java.util.Locale documentation. A locale can include a language, country, and variant. Language is the most common locale designation. Language can be extended with a country code. For instance, en\US, indicates American English. The name of the associated catalog is .en\US\mycat.xml. Variants are vendor or browser-specific and are used to introduce minor differences (for example, collation sequences) between two or more locales defined by either language or country.

## Guidelines for Naming Message Catalogs

Because the name of a message catalog file (without the .xml extension) is used to generate runtime class and property names, you should choose the name carefully.

Follow these guidelines for naming message catalogs:

- Do not choose a message catalog name that conflicts with the names of existing classes in the target package for which you are creating the message catalog.
- The message catalog name should only contain characters that are allowed in class names.
- Follow class naming standards.

For example, the resulting class names for a catalog named xyz.xml are xyzLogLocalizer and xyzLogger.

The following considerations also apply to message catalog files:

- Message IDs are generally six-character strings with leading zeros. Some interfaces also support integer representations.

---

**Note:** This only applies to log message catalogs. Simple text catalogs can have any string value.

---

- Java lets you group classes into a collection called a package. Oracle recommends that a package name be consistent with the name of the subsystem in which a particular catalog resides. Consistent naming makes OSGi imports easier to define.
- The log Localizer "classes" are actually ResourceBundle property files.

## Using Message Arguments

The message body, message detail, cause, and action sections of a log message can include message arguments, as described by `java.text.MessageFormat`. Make sure your message contents conform to the patterns specified by `java.text.MessageFormat`. Only the message body section in a simple text message can include arguments. Arguments are values that can be dynamically set at runtime. These values are passed to routines, such as printing out a message. A message can support up to 10 arguments, numbered 0-9. You can include any subset of these arguments in any text section of the message definition (Message Body, Message Detail, Probable Cause), although the message body must include all of the arguments. You insert message arguments into a message definition during development, and these arguments are replaced by the appropriate message content at runtime when the message is logged.

The following excerpt from an XML log message definition shows how you can use message arguments. The argument number must correspond to one of the arguments specified in the method attribute. Specifically, `{0}` with the first argument, `{1}` with the second, and so on. In [Example 23–10](#), `{0}` represents the file that cannot be opened, while `{1}` represents the file that will be opened in its place.

### Example 23–10 Message Arguments

```
<messagebody>Unable to open file, {0}. Opening {1}. All arguments must be in
body.</messagebody>

    <messagedetail> File, {0} does not exist. The server will restore the file
    contents from {1}, resulting in the use of default values for all future
    requests. </messagedetail>

    <cause>The file was deleted</cause>

    <action>If this error repeats then investigate unauthorized access to the
    file system.</action>
```

An example of a method attribute is as follows:

```
-method="logNoFile(String name, String path)"
```

The message example in [Example 23–10](#) expects two arguments, `{0}` and `{1}`:

- Both are used in the `<messagebody>`
- Both are used in the `<messagedetail>`
- Neither is used in `<cause>` or `<action>`

---

---

**Note:** A message can support up to 10 arguments, numbered 0-9. You can include any subset of these arguments in any text section of the message definition (message detail, cause, action), although the message body must include all of the arguments

---

---

In addition, the arguments are expected to be of `String` type, or representable as a `String` type. Numeric data is represented as `{n,number}`. Dates are supported as `{n,date}`. You must assign a severity level for log messages. Log messages are generated through the generated `Logger` methods, as defined by the method attribute.

## Message Catalog Formats

The catalog format for top-level and locale-specific catalog files is slightly different. The top-level catalogs define the textual messages for the base locale (by default, this is the English language). Locale-specific catalogs (for example, those translated to Spanish) only provide translations of text defined in the top-level version. Log message catalogs are defined differently from simple text catalogs.

**Log Message Catalog** [Example 23–11](#) shows a log message catalog, `MyUtilLog.xml`, containing one log message. This log message illustrates the usage of the `messagebody`, `messagedetail`, `cause`, and `action` elements.

### Example 23–11 Log Message Catalog

```
<?xml version="1.0"?>
<!DOCTYPE message_catalog PUBLIC "weblogic-message-catalog-dtd"
"http://www.bea.com/servers/wls90/dtd/msgcat.dtd">
<message_catalog
  l10n_package="programs.utils"
  i18n_package="programs.utils"
  subsystem="MYUTIL"
  version="1.0"
  baseid="600000"
  endid="600100"
  <log_message
    messageid="600001"
    severity="warning"
    method="logNoAuthorization(String arg0, java.util.Date arg1,int arg2)"
    <messagebody>
      Could not open file, {0} on {1,date} after {2,number} attempts.
    </messagebody>
    <messagedetail>
      The configuration for this application will be defaulted to
      factory settings. Custom configuration information resides
      in file, {0}, created on {1,date}, but is not readable.
    </messagedetail>
    <cause>
      The user is not authorized to use custom configurations. Custom
      configuration information resides in file, {0}, created on
      {1,date}, but is not readable.The attempt has been logged to
      the security log.
    </cause>
    <action>
      The user needs to gain appropriate authorization or learn to
      live with the default settings.
    </action>
  </log_message>
</message_catalog>
```

For more information, see [Appendix G, "Schema Reference: Message Catalog msgcat.dtd"](#).

**Simple Text Catalog** [Example 23–12](#) shows a simple text catalog, `MyUtilLabels.xml`, with one simple text definition.

### Example 23–12 Simple Text Catalog

```
<?xml version="1.0"?>
<!DOCTYPE message_catalog PUBLIC "weblogic-message-catalog-dtd"
"http://www.bea.com/servers/wls90/dtd/msgcat.dtd">
<message_catalog>
  l10n_package="programs.utils"
  i18n_package="programs.utils"
  subsystem="MYUTIL"
```

```
version="1.0"
<message>
  messageid="FileMenuTitle"
  <messagebody>
    File
  </messagebody>
</message>
</message_catalog>
```

For more information, see [Appendix G, "Schema Reference: Message Catalog msgcat.dtd"](#).

**Locale-Specific Catalog** [Example 23–13](#) shows a French translation of a message that is available in `.. \msgcat\fr\MyUtilLabels.xml`.

#### **Example 23–13** *Locale-Specific Catalog*

```
<?xml version="1.0"?>
<!DOCTYPE message_catalog PUBLIC
  "weblogic-locale-message-catalog-dtd"
  "http://www.bea.com/servers/wls90/dtd/l10n_msgcat.dtd">
<locale_message_catalog
  l10n_package="programs.utils"
  subsystem="MYUTIL"
  version="1.0">
  <message>
    <messageid="FileMenuTitle">
    <messagebody> Fichier </messagebody>
  </message>
</locale_message_catalog>
```

When entering text in the `messagebody`, `messagedetail`, `cause`, and `action` elements, you must use a tool that generates valid Unicode Transformation Format-8 (UTF-8) characters, and have appropriate keyboard mappings installed. UTF-8 is an efficient encoding of Unicode character-strings that optimizes the encoding ASCII characters. Message catalogs always use UTF-8 encoding.

For more information, see [Appendix H, "Schema Reference: Locale Message Catalog l10n\\_msgcat.dtd"](#).

### **Message Catalog Localization**

Catalog subdirectories are named after lowercase, two-letter ISO 639 language codes (for example, `ja` for Japanese and `fr` for French). You can find supported language codes in the `java.util.Locale` Javadoc.

Variations to language codes are achievable through the use of uppercase, two-letter ISO 3166 country codes and variants, each of which are subordinate to the language code. The generic syntax is `lang\country\variant`.

For example, `zh` is the language code for Chinese. `CN` is a country code for simplified Chinese, whereas `TW` is the country code for traditional Chinese. Therefore `zh\CN` and `zh\TW` are two distinct locales for Chinese.

Variants are of use when, for instance, there is a functional difference in platform vendor handling of specific locales. Examples of vendor variants are `WIN`, `MAC`, and `POSIX`. There may be two variants used to further qualify the locale. In this case, the variants are separated with an underscore (for example, `Traditional_Mac` as opposed to `Modern_MAC`).

---

---

**Note:** Language, country, and variants are all case sensitive.

---

---

A fully-qualified locale would look like `zh\TW\WIN`, identifying traditional Chinese on a Win32 platform.

Message catalogs to support the above locale would involve the following files:

- `\*.xml` - default catalogs
- `\zh\*.xml` - Chinese localizations
- `\zh\TW\*.xml` - Traditional Chinese localizations
- `\zh\TW\WIN\*.xml` - Traditional Chinese localizations for Win32 code sets

Specific localizations do not need to cover all messages defined in parent localizations.

## How to Parse a Message Catalog to Generate Logger and TextFormatter Classes for Localization

After you create your message catalog XML file, you can use the `weblogic.i18ngen` utility to create `Logger` and `TextFormatter` classes.

use the `weblogic.i18ngen` utility to parse message catalogs (XML files) to produce `Logger` and `TextFormatter` classes used for localizing the text in log messages. The utility creates or updates the `i18n_user.properties` properties file, which is used to load the message ID lookup class hashtable `weblogic.i18n.L10nLookup`.

Any errors, warnings, or informational messages are sent to `stderr`.

In order for user catalogs to be recognized, the `i18n_user.properties` file must reside in a directory identified in the Oracle Event Processing server classpath.

Oracle recommends that the `i18n_user.properties` file reside in the Oracle Event Processing server classpath. If the `i18n_user.properties` file is in `targetdirectory`, then `targetdirectory` should be in the Oracle Event Processing server classpath.

### To parse a message catalog to generate `Logger` and `TextFormatter` classes:

1. Create your message catalog XML file.  
See [Section , "Using Message Catalogs With Oracle Event Processing Server"](#).
2. Set up your development environment.  
See [Section , "Setting Your Development Environment."](#)
3. Execute the `weblogic.i18ngen` utility using the following syntax:

```
java weblogic.i18ngen [options] [filelist]
```

Where:

- *options*: see [Table 23–6](#).
- *filelist*: Process the files and directories in this list of files. If directories are listed, the command processes all XML files in the listed directories. The names of all files must include an XML suffix. All files must conform to the `msgcat.dtd` syntax. `weblogic.i18ngen` prints the fully-qualified list of names (Java source) to `stdout` for those files actually generated.

**Table 23–6** *weblogic.i18ngen* Utility Options

Option	Description
<code>-build</code>	Generates all necessary files and compiles them. The <code>-build</code> option combines the <code>-i18n</code> , <code>-l10n</code> , <code>-keepgenerated</code> , and <code>-compile</code> options.

**Table 23–6 (Cont.) weblogic.i18ngen Utility Options**

Option	Description
-d <i>targetdirectory</i>	Specifies the root directory to which generated Java source files are targeted. User catalog properties are placed in <code>i18n_user.properties</code> , relative to the designated <i>targetdirectory</i> . Files are placed in appropriate directories based on the <code>i18n_package</code> and <code>i10n_package</code> values in the corresponding message catalog. The default target directory is the current directory. This directory is created as necessary.  If this argument is omitted, all classes are generated in the current directory, without regard to any class hierarchy described in the message catalog.
-n	Parse and validate, but do not generate classes.
-keepgenerated	Keep generated Java source (located in the same directory as the class files).
-ignore	Ignore errors.
-i18n	Generates internationalizers (for example, <code>Loggers</code> and <code>TextFormatters</code> ).
-i10n	Generates localizers (for example, <code>LogLocalizers</code> and <code>TextLocalizers</code> ).
-compile	Compiles generated Java files using the current CLASSPATH. The resulting classes are placed in the directory identified by the -d option. The resulting classes are placed in the same directory as the source.  Errors detected during compilation generally result in no class files or properties file being created. <code>i18ngen</code> exits with a bad exit status.
-nobuild	Parse and validate only.
-debug	Debugging mode.
-dates	Causes <code>weblogic.i18ngen</code> to update message timestamps in the catalog. If the catalog is writable and timestamps have been updated, the catalog is rewritten.

---

**Note:** Utilities can be run from any directory, but if files are listed on the command line, then their path is relative to the current directory.

---

4. Translate your log messages and generate the required localized resource bundles.
5. Ensure that the `i18n_user.properties` file is in the Oracle Event Processing server classpath.
6. Import the following packages in your Oracle Event Processing application:
  - `weblogic.i18n.logging`
  - `weblogic.logging`
7. Assemble and deploy your application, including your log message resource bundles.

## Deploying Oracle Event Processing Applications

After you assemble your Oracle Event Processing application, you deploy it to an Oracle Event Processing server domain.

This section describes:

- [Section , "How to Deploy an Oracle Event Processing Application Using Oracle Event Processing IDE for Eclipse"](#)
- [Section , "How to Deploy an Oracle Event Processing Application Using Oracle Event Processing Visualizer"](#)
- [Section , "How to Deploy an Oracle Event Processing Application Using the Deployer Utility"](#)

For more information, see:

- "Deploying an Application to an Oracle Event Processing Standalone-Server Domain" in the *Oracle Fusion Middleware Administrator's Guide for Oracle Event Processing*
- "Deploying an Oracle Event Processing Application to a Multi-Server Domain" in the *Oracle Fusion Middleware Administrator's Guide for Oracle Event Processing*

## How to Deploy an Oracle Event Processing Application Using Oracle Event Processing IDE for Eclipse

You can deploy an Oracle Event Processing application using Oracle Event Processing IDE for Eclipse.

Using the Oracle Event Processing IDE for Eclipse, you can deploy an application to either a stand-alone or multi-server domain.

---

---

**Note:** If you are using foreign stages, beware of the rules governing deployment and redeployment of dependent stages as [Section , "Assembling Applications With Foreign Stages"](#) describes.

---

---

### To deploy an Oracle Event Processing application using Oracle Event Processing IDE for Eclipse:

1. Assemble your Oracle Event Processing application.  
See [Section , "Assembling an Oracle Event Processing Application."](#)
2. Use the Oracle Event Processing IDE for Eclipse to deploy your application.  
See [Section , "How to Deploy an Application to an Oracle Event Processing Server"](#).

## How to Deploy an Oracle Event Processing Application Using Oracle Event Processing Visualizer

The simplest way to deploy an Oracle Event Processing application to an Oracle Event Processing server domain is to use the Oracle Event Processing Visualizer.

Using the Oracle Event Processing Visualizer, you can deploy an application to either a stand-alone or multi-server domain.

---

---

**Note:** If you are using foreign stages, beware of the rules governing deployment and redeployment of dependent stages as [Section , "Assembling Applications With Foreign Stages"](#) describes.

---

---

### To deploy an Oracle Event Processing application using Oracle Event Processing Visualizer:

1. Assemble your Oracle Event Processing application.  
See [Section , "Assembling an Oracle Event Processing Application."](#)
2. Start the Oracle Event Processing Visualizer.  
See [Section , "How to Start the Oracle Event Processing Visualizer from Oracle Event Processing IDE for Eclipse"](#).

3. Use the Oracle Event Processing Visualizer to deploy your application.

See "Deploying an Application" in the *Oracle Fusion Middleware Visualizer User's Guide for Oracle Event Processing*.

## How to Deploy an Oracle Event Processing Application Using the Deployer Utility

The following procedure describes how to deploy an application to Oracle Event Processing 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 Fusion Middleware Administrator's Guide for Oracle Event Processing*.

---

---

**Note:** If you are using foreign stages, beware of the rules governing deployment and redeployment of dependent stages as [Section](#) , "[Assembling Applications With Foreign Stages](#)" describes.

---

---

### To deploy an Oracle Event Processing application using the Deployer utility:

1. Assemble your Oracle Event Processing application.  
See [Section](#) , "[Assembling an Oracle Event Processing Application](#)."
2. Open a command window and set your environment as described in [Section](#) , "[Setting Your Development Environment](#)."
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 Event Processing 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 Fusion Middleware Administrator's Guide for Oracle Event Processing*.

---

---

4. Be sure you have configured Jetty for the Oracle Event Processing instance to which you are deploying your application.

For more information, see "Configuring Jetty for Oracle Event Processing" in the *Oracle Fusion Middleware Administrator's Guide for Oracle Event Processing*.

5. In the command window, run the Deployer utility using the following syntax to install your application (in practice, the command should be on one line):

```
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 Event Processing is running.
- `port` refers to the port number to which Oracle Event Processing listens; default value is 9002.

This port is specified in the `DOMAIN_DIR/config/config.xml` file that describes your Oracle Event Processing 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 Event Processing administrator.
- *password* refers to the password of the Oracle Event Processing administrator.
- *application\_jar\_file* refers to your application JAR file, assembled into an OSGi bundle as described in [Section , "Assembling an Oracle Event Processing Application."](#) This file must be located on the same computer from which you execute the Deployer utility.

For example, if Oracle Event Processing 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 (in practice, the command should be on one line):

```
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 Event Processing 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 Fusion Middleware Administrator's Guide for Oracle Event Processing*.

---

**Note:** You may only deploy to a group if the server is part of a multi-server domain (that is, if clustering is enabled). You may not deploy to a group if the server is part of a standalone-server domain (that is, if clustering is disabled). For more information, see "Overview of Oracle Event Processing Multi-Server Domain Administration" in the *Oracle Fusion Middleware Administrator's Guide for Oracle Event Processing*.

---

Oracle Event Processing 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 , "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.



---

## Developing Applications for High Availability

This chapter introduces Oracle Event Processing components and design patterns that you can use to increase the availability of your Oracle Event Processing applications, along with how to configure high availability for your Oracle Event Processing application to provide the quality of service you require.

This chapter includes the following sections:

- [Understanding High Availability](#)
- [Configuring High Availability](#)

### Understanding High Availability

This section introduces Oracle Event Processing components and design patterns that you can use to increase the availability of your Oracle Event Processing applications.

This section includes the following sections:

- [Section , "High Availability Architecture"](#)
- [Section , "Choosing a Quality of Service"](#)
- [Section , "Designing an Oracle Event Processing Application for High Availability"](#)

### High Availability Architecture

Like any computing resource, Oracle Event Processing servers can be subject to both hardware and software faults that can lead to data- or service-loss. Oracle Event Processing high availability options seek to mitigate both the likelihood and the impact of such faults.

Oracle Event Processing supports an active-standby high availability architecture. This approach has the advantages of high performance, simplicity, and short failover time.

An Oracle Event Processing application that needs to be highly available is deployed to a group of two or more Oracle Event Processing server instances running in an Oracle Event Processing multi-server domain. Oracle Event Processing will automatically choose one server in the group to be the active primary. The remaining servers become active secondaries.

The primary and secondary servers are all configured to receive the same input events and process them in parallel but only the primary server outputs events to the Oracle Event Processing application client. Depending on the quality of service you choose, the secondary servers buffer their output events using in-memory queues and the primary server keeps the secondary servers up to date with which events the primary has already output.

Figure 24–1 shows a typical configuration.

**Figure 24–1 Oracle Event Processing High Availability: Primary and Secondary Servers**



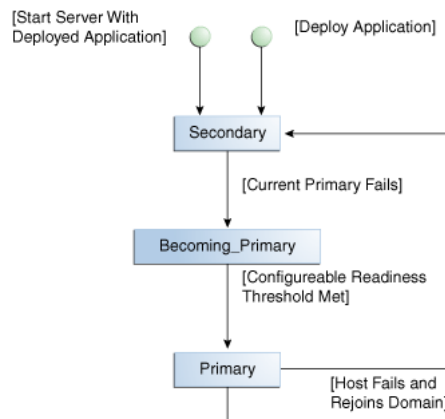
This section describes:

- Section , "High Availability Lifecycle and Failover"
- Section , "Deployment Group and Notification Group"
- Section , "High Availability Components"
- Section , "High Availability and Scalability"
- Section , "High Availability and Oracle Coherence"

**High Availability Lifecycle and Failover**

Figure 24–2 shows a state diagram for the Oracle Event Processing high availability lifecycle. In this diagram, the state names (SECONDARY, BECOMING\_PRIMARY, and PRIMARY) correspond to the Oracle Event Processing high availability adapter RuntimeMBean method getState return values. These states are specific to Oracle Event Processing.

**Figure 24–2 Oracle Event Processing High Availability Lifecycle State Diagram**



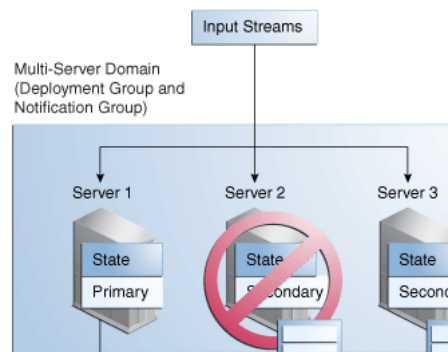
It is not possible to specify the server that will be the initial primary. Initially, the first server in the multi-server domain to start up becomes the primary so by starting servers in a particular order, you can influence primary selection. There is no way to force a particular, running server to become the primary. If a primary fails, and then comes back up, it will not automatically become the primary again unless the current primary fails causing a failover.

This section describes the Oracle Event Processing high availability lifecycle in more detail, including:

- [Section , "Secondary Failure"](#)
- [Section , "Primary Failure and Failover"](#)
- [Section , "Rejoining the High Availability Multi-Server Domain"](#)

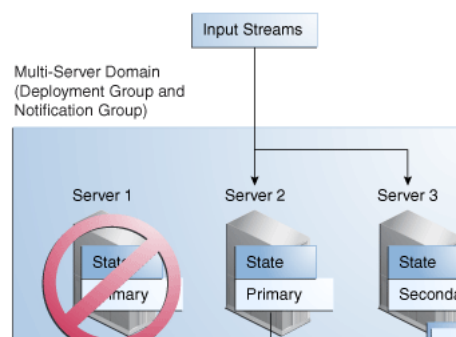
**Secondary Failure** In general, when a secondary server fails, there is no effect on Oracle Event Processing application operation as [Figure 24–3](#) shows. Regardless of the quality of service you choose, there are no missed or duplicate events.

**Figure 24–3 Secondary Failure**



**Primary Failure and Failover** However, when a primary server fails, as [Figure 24–4](#) shows, Oracle Event Processing performs a failover that may cause missed or duplicate events, depending on the quality of service you choose.

**Figure 24–4 Primary Failure and Failover**



During failover, Oracle Event Processing automatically selects a new primary and the new primary transitions from the `SECONDARY` state to the `BECOMING_PRIMARY` state. Depending on the quality of service you choose, the new primary will not transition to `PRIMARY` state until a configurable readiness threshold is met. For details, see the specific quality of service option in [Section , "Choosing a Quality of Service"](#).

**Rejoining the High Availability Multi-Server Domain** When a new Oracle Event Processing server is added to an Oracle Event Processing high availability multi-server domain or an existing failed server restarts, the server will not have fully joined the Oracle Event Processing high availability deployment and notification groups until all applications deployed to it have fully joined. The type of application determines when it can be said to have fully joined.

If the application must generate exactly the same sequence of output events as existing secondaries (a Type 1 application), then it must be able to rebuild its internal state by processing input streams for some finite period of time (the `warm-up-window-length` period). This `warm-up-window-length` time determines the minimum time it will take for the application to fully join the Oracle Event Processing high availability deployment and notification groups.

If the application does not need to generate exactly the same sequence of output events as existing secondaries (a Type 2 application), then it does not require a `warm-up-window-length` time and will fully join the Oracle Event Processing high availability deployment and notification groups as soon as it is deployed.

For more information, see [Section , "Choose an Adequate warm-up-window-length Time"](#).

### Deployment Group and Notification Group

All the servers in the multi-server domain belong to the same deployment group: this is the group to which you deploy an application. For the purposes of Oracle Event Processing high availability, you must deploy the same application to all the servers in this group.

By default, all the servers in the multi-server domain also belong to the same notification group. The servers listen to the notification group for membership notifications that indicate when a server has failed (and exited the group) or resumed operation (and rejoined the group), as well as for synchronization notifications from the primary.

If you need to scale your Oracle Event Processing high availability application, you can use the `ActiveActiveGroupBean` to define a notification group that allows two or more servers to function as a primary server unit while retaining the convenience of a single deployment group that spans all servers (primaries and secondaries).

You must use Oracle Coherence-based clustering to create the multi-server domain deployment group. You may use either default groups or custom groups.

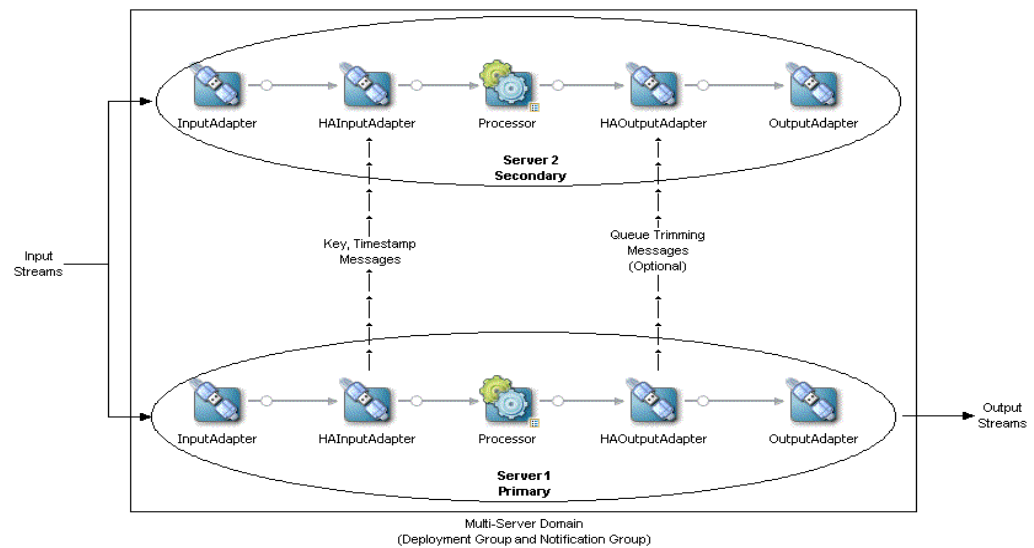
For more information, see:

- [Section , "High Availability and Scalability"](#)
- [Section , "High Availability and Oracle Coherence"](#)
- "How to Create an Oracle Event Processing Multi-Server Domain With Default Groups Using Oracle Coherence" in the *Oracle Fusion Middleware Administrator's Guide for Oracle Event Processing*.
- "How to Create an Oracle Event Processing Multi-Server Domain With Custom Groups Using Oracle Coherence" in the *Oracle Fusion Middleware Administrator's Guide for Oracle Event Processing*.

### High Availability Components

To implement Oracle Event Processing high availability options, you configure your Event Processing Network (EPN) with a high availability input adapter after each input adapter and a high availability output adapter before each output adapter.

[Figure 24–5](#) shows a typical EPN with all possible high availability adapters in place.

**Figure 24–5 High Availability Adapters in the EPN**

**Note:** For simplicity, Figure 24–5 does not show channels and shows only one processor. However, the EPN may be arbitrarily complex with multiple input streams and output streams, channels, multiple processors, event beans, and so on. The only restriction is that each input adapter must be followed by a high availability input adapter and each output adapter must be preceded by a high availability output adapter. Similarly, for simplicity, a multi-server domain of only two Oracle Event Processing servers is shown but you may have an arbitrary number of secondary servers.

The optional high availability input adapter in the primary communicates with the corresponding high availability input adapters in each secondary to normalize event timestamps.

Oracle Event Processing high availability provides one type of high availability input adapter. See [Section , "High Availability Input Adapter"](#).

The high availability output adapter in the primary is responsible for outputting events to the output streams that connect the Oracle Event Processing application to its downstream client. The high availability output adapter in the primary also communicates with the corresponding high availability output adapters in each secondary, and, depending on the high availability quality of service you choose, may instruct the secondary output adapters to trim their in-memory queues of output events.

Oracle Event Processing high availability provides the following high availability output adapters:

- [Section , "Buffering Output Adapter"](#)
- [Section , "Broadcast Output Adapter"](#)
- [Section , "Correlating Output Adapter"](#)

Oracle Event Processing high availability also provides a notification groups Spring bean to increase scalability in JMS applications. See [Section , "ActiveActiveGroupBean"](#).

Which adapter you choose is determined by the high availability quality of service you choose. See [Section , "Choosing a Quality of Service"](#).

**High Availability Input Adapter** The optional Oracle Event Processing high availability input adapter on the primary Oracle Event Processing server assigns a time (in nanoseconds) to events as they arrive at the adapter and forwards the time values assigned to events to all secondary servers. This ensures that all servers running the application use a consistent time value (and generate the same results) and avoids the need for distributed clock synchronization.

Since a time value is assigned to each event before the event reaches any downstream channels in the EPN, downstream channels should be configured to use application time so that they do not assign a new time value to events as they arrive at the channel.

Input events must have a key that uniquely identifies each event in order to use this adapter.

You can configure the Oracle Event Processing high availability input adapter to send heartbeat events.

The Oracle Event Processing high availability input adapter is applicable to all high availability quality of service options. However, because the high availability input adapter increases performance overhead, it is not appropriate for some high availability quality of service options (such as [Section , "Simple Failover"](#) and [Section , "Simple Failover with Buffering"](#)). For these options, you should instead consider using application time with some incoming event property.

For more information, see:

- [Section , "Light-Weight Queue Trimming"](#)
- [Section , "Precise Recovery with JMS"](#)
- [Section , "How to Configure the High Availability Input Adapter"](#).

**Buffering Output Adapter** The Oracle Event Processing high availability buffering output adapter implements a buffered queue trimming strategy. The buffer is a sliding window of output events from the stream. The size of the window is measured in milliseconds.

The Oracle Event Processing high availability buffering output adapter is applicable to simple failover and simple failover with buffering high availability quality of service options.

For more information, see:

- [Section , "Simple Failover"](#)
- [Section , "Simple Failover with Buffering"](#)
- [Section , "How to Configure the Buffering Output Adapter"](#).

**Broadcast Output Adapter** The Oracle Event Processing high availability broadcast output adapter implements a distributed queue trimming strategy. The active primary instance broadcasts messages to the active secondary instances in the notification group telling them when to trim their local representation of the queue.

The Oracle Event Processing high availability broadcast output adapter is applicable to the light-weight queue trimming high availability quality of service option.

For more information, see:



- [Section , "Light-Weight Queue Trimming"](#)
- [Section , "How to Configure the Broadcast Output Adapter"](#).

**Correlating Output Adapter** The Oracle Event Processing high availability correlating output adapter correlates two event streams, usually from JMS. This adapter correlates an inbound buffer of events with a second source of the same event stream, outputting the buffer if correlation fails after a configurable time interval. Correlated events are trimmed from the queue. Correlated events are assumed to be in-order.

The Oracle Event Processing high availability correlating output adapter is applicable to precise recovery with JMS high availability quality of service option.

For more information, see:

- [Section , "Precise Recovery with JMS"](#)
- [Section , "How to Configure the Correlating Output Adapter"](#).

**ActiveActiveGroupBean** The

`com.oracle.cep.cluster.hagroups.ActiveActiveGroupBean` is a Spring bean that allows you to partition an input stream from a JMS input adapter.

This component is applicable to precise recovery with JMS high availability quality of service only. However, it can also be used without high availability to increase Oracle Event Processing application scalability.

For more information, see:

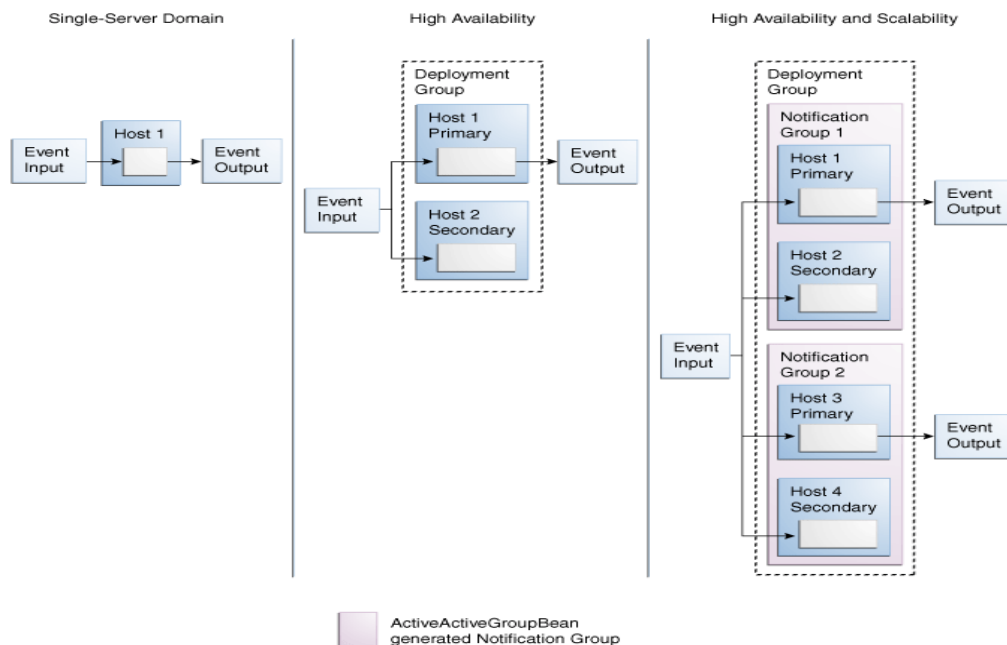
- [Section , "High Availability and Scalability"](#)
- [Section , "Precise Recovery with JMS"](#)
- [Section , "ActiveActiveGroupBean"](#)

### High Availability and Scalability

If you need to scale your Oracle Event Processing high availability application, you can use the `ActiveActiveGroupBean` to define a notification group that allows two or more servers to function as a high availability unit while retaining the convenience of a single deployment group that spans all servers (primaries and secondaries).

[Figure 24–6](#) shows three Oracle Event Processing application scenarios progressing from the simplest configuration, to high availability, and then to both high availability and scalability.

**Figure 24–6 High Availability and Scalability**



Most applications begin in a single-server domain without high availability. In this, the simplest scenario, an Oracle Event Processing application running on one Oracle Event Processing server processes an input event stream and produces output events.

In the high availability scenario, the Oracle Event Processing application has been configured to use Oracle Event Processing high availability options. This application is deployed to the deployment group of a multi-server domain composed of two servers. In this scenario, only the primary server outputs events.

In the high availability and scalability scenario, the Oracle Event Processing high availability application has been configured to use the `ActiveActiveGroupBean` to define notification groups. Each notification group contains two or more Oracle Event Processing servers that function as a single, high availability unit. In this scenario, only the primary server in each notification group outputs events. Should the primary server in a notification group go down, an Oracle Event Processing high availability fail over occurs and a secondary server in that notification group is declared the new primary and resumes outputting events according to the Oracle Event Processing high availability quality of service you configure.

For more information, see:

- [Section , "ActiveActiveGroupBean"](#)
- [Section , "How to Configure Scalability in a JMS Application With Oracle Event Processing High Availability"](#)

**High Availability and Oracle Coherence**

Oracle Event Processing high availability options depend on Oracle Coherence. You cannot implement Oracle Event Processing high availability options without Oracle Coherence.

When considering performance tuning, be sure to evaluate your Oracle Coherence configuration in addition to your Oracle Event Processing application.

For more information, see:

- [Section , "Oracle Coherence Performance Tuning Options"](#)
- "Configuring the Oracle Coherence Cluster" in the *Oracle Fusion Middleware Administrator's Guide for Oracle Event Processing*
- *Oracle Coherence Developer's Guide* at [http://download.oracle.com/docs/cd/E15357\\_01/coh.360/e15723/toc.htm](http://download.oracle.com/docs/cd/E15357_01/coh.360/e15723/toc.htm)

## Choosing a Quality of Service

Using Oracle Event Processing high availability, you may choose any of the quality of service options that [Table 24-1](#) lists. Choose the quality of service option that suits your application's tolerance for missed and duplicate events as well as expected event throughput. Note that primary and secondary server hardware requirements increase as the quality of service becomes more precise.

**Table 24-1 Oracle Event Processing High Availability Quality of Service**

High Availability Option	Missed Events?	Duplicate Events?	Performance Overhead
<a href="#">Section , "Simple Failover"</a>	Yes (many)	Yes (few)	Negligible
<a href="#">Section , "Simple Failover with Buffering"</a>	Yes (few) <sup>1</sup>	Yes (many)	Low
<a href="#">Section , "Light-Weight Queue Trimming"</a>	No	Yes (few)	Low-Medium <sup>2</sup>
<a href="#">Section , "Precise Recovery with JMS"</a>	No	No	High

<sup>1</sup> If you configure a big enough buffer then there will be no missed events.

<sup>2</sup> The performance overhead is tunable. You can adjust the frequency of trimming to reduce the overhead, but incur a higher number of duplicates at failover.

### Simple Failover

This high availability quality of service is characterized by the lowest performance overhead (fastest recovery time) and the least data integrity (both missed events and duplicate events are possible during failover).

The primary server outputs events and secondary servers simply discard their output events; they do not buffer output events. If the current active primary fails, a new active primary is chosen and begins sending output events once it is notified.

During failover, many events may be missed or duplicated by the new primary depending on whether it is running ahead of or behind the old primary, respectively.

During the failover window, events may be missed. For example, if you are processing 100 events per second and failover takes 10 s then you miss 1000 events

The new primary enters the PRIMARY state immediately. There is no configurable readiness threshold that must be met before the new primary transitions out of the BECOMING\_PRIMARY state.

When an Oracle Event Processing server rejoins the multi-server domain, it is immediately available as a secondary.

To implement this high availability quality of service, you configure your EPN with a high availability buffering output adapter (with a sliding window of size zero) before each output adapter. To reduce performance overhead, rather than use a high availability input adapter, use application time with some incoming event property.

For more information, see [Section , "How to Configure Simple Failover"](#).

### Simple Failover with Buffering

This high availability quality of service is characterized by a low performance overhead (faster recovery time) and increased data integrity (no missed events but many duplicate events are possible during failover).

The primary server outputs events and secondary servers buffer their output events. If the current active primary fails, a new active primary is chosen and begins sending output events once it is notified.

During the failover window, events may be missed. For example, if you are processing 100 events per second and failover takes 10 s then you miss 1000 events. If the secondary buffers are large, a significant number of duplicates may be output. On the other hand, a larger buffer reduces the chances of missed messages.

When an Oracle Event Processing server rejoins the multi-server domain, if your application is an Oracle Event Processing high availability Type 1 application (the application must generate exactly the same sequence of output events as existing secondaries), it must wait the `warm-up-window-length` time you configure for the Oracle Event Processing high availability output adapter before it is available as a secondary.

To implement this high availability quality of service, you configure your EPN with a high availability buffering output adapter (with a sliding window of size greater than zero) before each output adapter. To reduce performance overhead, rather than use a high availability input adapter, use application time with some incoming event property.

For more information, see:

- [Section , "Choose an Adequate warm-up-window-length Time"](#)
- [Section , "How to Configure Simple Failover With Buffering"](#)

### Light-Weight Queue Trimming

This high availability quality of service is characterized by a low performance overhead (faster recovery time) and increased data integrity (no missed events but a few duplicate events are possible during failover).

The active primary communicates to the secondaries the events that it has actually processed. This enables the secondaries to trim their buffer of output events so that it contains only those events that have not been sent by the primary at a particular point in time. Because events are only trimmed after they have been sent by the current primary, this allows the secondary to avoid missing any output events when there is a failover.

The frequency with which the active primary sends queue trimming messages to active secondaries is configurable:

- Every  $n$  events ( $n > 0$ )  
This limits the number of duplicate output events to at most  $n$  events at failover.
- Every  $n$  milliseconds ( $n > 0$ )

The queue trimming adapter requires a way to identify events consistently among the active primary and secondaries. The recommended approach is to use application time to identify events, but any key value that uniquely identifies events will do.

The advantage of queue trimming is that output events are never lost. There is a slight performance overhead at the active primary, however, for sending the trimming

messages that need to be communicated and this overhead increases as the frequency of queue trimming messages increases.

During failover, the new primary enters the `BECOMING_PRIMARY` state and will not transition into the `PRIMARY` state until its event queue (that it was accumulating as a secondary) has been flushed. During this transition, new input events are buffered and some duplicate events may be output.

When an Oracle Event Processing server rejoins the multi-server domain, if your application is an Oracle Event Processing high availability Type 1 application (an application that must generate exactly the same sequence of output events as existing secondaries), it must wait the `warm-up-window-length` time you configure for the Oracle Event Processing high availability output adapter before it is available as a secondary.

To implement this high availability quality of service, you configure your EPN with a high availability input adapter after each input adapter and a high availability broadcast output adapter before each output adapter.

For more information, see [Section , "How to Configure Light-Weight Queue Trimming"](#).

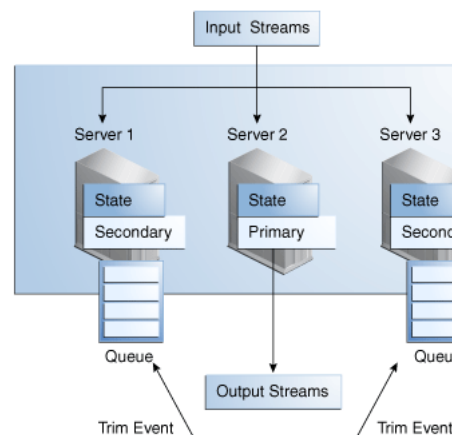
### Precise Recovery with JMS

This high availability quality of service is characterized by a high performance overhead (slower recovery time) and maximum data integrity (no missed events and no duplicate events during failover).

This high availability quality of service is compatible with only JMS input and output adapters.

In this high availability quality of service, we are not concerned with transactional guarantees along the event path for a single-server but in guaranteeing a single output from a set of servers. To achieve this, secondary servers listen, over JMS, to the event stream being published by the primary. As [Figure 24–7](#) shows, this incoming event stream is essentially a source of reliable queue-trimming messages that the secondaries use to trim their output queues. If JMS is configured for reliable delivery we can be sure that the stream of events seen by the secondary is precisely the stream of events output by the primary and thus failover will allow the new primary to output precisely those events not delivered by the old primary.

**Figure 24–7** *Precise Recovery with JMS*



During failover, the new primary enters the `BECOMING_PRIMARY` state and will not transition into the `PRIMARY` state its event queue (that it was accumulating as a secondary) has been flushed. During this transition, new input events are buffered and no duplicate events are output.

When an Oracle Event Processing server rejoins the multi-server domain, if your application is an Oracle Event Processing high availability Type 1 application (the application must generate exactly the same sequence of output events as existing secondaries), it must wait the `warm-up-window-length` time you configure for the Oracle Event Processing high availability output adapter before it is available as a secondary.

To implement this high availability quality of service, you configure your EPN with a high availability input adapter after each input adapter and a high availability correlating output adapter before each output adapter.

To increase scalability, you can also use the cluster groups bean with high availability quality of service.

For more information, see:

- [Section , "How to Configure Precise Recovery With JMS"](#)
- [Section , "Configuring Scalability With the ActiveActiveGroupBean"](#)

## Designing an Oracle Event Processing Application for High Availability

Although you can implement Oracle Event Processing high availability declaratively, to fully benefit from the high availability quality of service you choose, you must design your Oracle Event Processing application to take advantage of the high availability options that Oracle Event Processing provides.

When designing your Oracle Event Processing application for high availability, consider the following:

- [Section , "Primary Oracle Event Processing High Availability Use Case"](#)
- [Section , "High Availability Design Patterns"](#)
- [Section , "Oracle CQL Query Restrictions"](#)

### Primary Oracle Event Processing High Availability Use Case

You can adapt Oracle Event Processing high availability options to various Oracle Event Processing application designs but in general, Oracle Event Processing high availability is designed for the following use case:

- An application receives input events from one or more external systems.
- The external systems are publish-subscribe style systems that allow multiple instances of the application to connect simultaneously and receive the same stream of messages.
- The application does not update any external systems in a way that would cause conflicts should multiple copies of the application run concurrently.
- The application sends output events to an external downstream system. Multiple instances of the application can connect to the downstream system simultaneously, although only one instance of the application is allowed to send messages at any one time.

Within these constraints, the following different cases are of interest:

- The application is allowed to skip sending some output events to the downstream system when there is a failure. Duplicates are also allowed.  
For this case, the following Oracle Event Processing high availability quality of service options are applicable:
  - [Section , "Simple Failover"](#)
- The application is allowed to send duplicate events to the downstream system, but must not skip any events when there is a failure.  
For this case, the following Oracle Event Processing high availability quality of service options are applicable:
  - [Section , "Simple Failover with Buffering"](#)
  - [Section , "Light-Weight Queue Trimming"](#)
- The application must send exactly the same stream of messages/events to the downstream system when there is a failure, modulo a brief pause during which events may not be sent when there is a failure.  
For this case, the following Oracle Event Processing high availability quality of service options are applicable
  - [Section , "Precise Recovery with JMS"](#)

### High Availability Design Patterns

When designing your Oracle Event Processing application for use with Oracle Event Processing high availability options, observe the following design patterns:

- [Section , "Select the Minimum High Availability Your Application can Tolerate"](#)
- [Section , "Use Oracle Event Processing High Availability Components at All Ingress and Egress Points"](#)
- [Section , "Only Preserve What You Need"](#)
- [Section , "Limit Oracle Event Processing Application State"](#)
- [Section , "Choose an Adequate warm-up-window-length Time"](#)
- [Section , "Ensure Applications are Idempotent"](#)
- [Section , "Source Event Identity Externally"](#)
- [Section , "Understand the Importance of Event Ordering"](#)
- [Section , "Write Oracle CQL Queries with High Availability in Mind"](#)
- [Section , "Avoid Coupling Servers"](#)
- [Section , "Plan for Server Recovery"](#)

**Select the Minimum High Availability Your Application can Tolerate** Be sure that the extra cost of precise recovery (per-node throughput decrease) is actually necessary for your application.

**Use Oracle Event Processing High Availability Components at All Ingress and Egress Points** You must use an Oracle Event Processing high availability input adapter after each regular input adapter and you must use an Oracle Event Processing high availability output adapter before each regular output adapter.

**Only Preserve What You Need** Most Oracle Event Processing systems are characterized by a large number of raw input events being queried to generate a smaller number of

“enriched” events. In general it makes sense to only try and preserve these enriched events – both because there are fewer of them and because they are more valuable.

**Limit Oracle Event Processing Application State** Oracle Event Processing systems allow you to query windows of events. It can be tempting to build systems using very large windows, but this increases the state that needs to be rebuilt when failure occurs. In general it is better to think of long-term state as something better kept in stable storage, such as a distributed cache or a database – since the high availability facilities of these technologies can be appropriately leveraged.

**Choose an Adequate warm-up-window-length Time** When a new Oracle Event Processing server is added to an Oracle Event Processing high availability multi-server domain or an existing failed server restarts, the server will not have fully joined the Oracle Event Processing high availability deployment and notification groups until all applications deployed to it have fully joined. The type of application determines when it can be said to have fully joined.

Oracle Event Processing high availability applications can be described as Type 1 or Type 2 applications as [Table 24–2](#) shows.

**Table 24–2 Oracle Event Processing High Availability Application Types**

Application Type	Must generate exactly the same sequence of output events?	Must be able to rebuild internal state by processing input streams within a finite period of time?	Must wait this period of time before it has fully joined?
Type 1	Yes	Yes	Yes
Type 2	No	No	No

For more information, see [Section , "Rejoining the High Availability Multi-Server Domain"](#).

**Type 1 Applications** A Type 1 application requires the new secondary to generate exactly the same sequence of output events as existing secondaries once it fully joins the Oracle Event Processing high availability deployment and notification groups.

It is a requirement that a Type 1 application be able to rebuild its internal state by processing its input streams for some finite period of time (warm-up-window-length time), after which it generates exactly the same stream of output events as other secondaries running the application.

The warm-up-window-length time is configured on an Oracle Event Processing high availability output adapter. The warm-up-window-length time length is specified in terms of seconds or minutes. For example, if the application contains Oracle CQL queries with range-based windows of 5, 7, and 15 minutes then the minimum warm-up-window-length time is 15 minutes (the maximum range-based window size). Oracle recommends that the maximum window length be padded with a few minutes time, as well, to absolutely ensure that the necessary state is available. So, in the previous example 17 minutes or even 20 minutes would be a good length for the warm-up-window-length time.

The Oracle Event Processing server uses system time during the warm-up-window-length time period, so it is not directly correlated with the application time associated with events being processed.

Type 1 applications must only be interested in events that occurred during a finite amount of time. All range-based Oracle CQL windows must be shorter than the warm-up-window-length time and tuple-based windows must also be qualified by



time. For example, the application should only care about the last 10 events if they occurred within the last five minutes. Applications that do not have this property cannot be Type 1 applications and cannot use the `warm-up-window-length` period. For example, an application that uses an tuple-based partitioned window that has no time qualification cannot use the `warm-up-window-length` period, since an arbitrary amount of time is required to rebuild the state of the window.

If a Type 1 application uses the Oracle Event Processing high availability broadcast output adapter, it may trim events using a unique application-specific key, or a monotonic key like application time. Trimming events using application time is encouraged as it is more robust and less susceptible to bugs in the application that may cause an output event to fail to be generated.

For more information, see:

- [Section , "Oracle CQL Query Restrictions"](#)
- [Section , "Buffering Output Adapter"](#)
- [Section , "Broadcast Output Adapter"](#)
- [Section , "Correlating Output Adapter"](#)

**Type 2 Applications** A Type 2 application does not require the new secondary to generate exactly the same sequence of output events as existing secondaries once it fully joins the Oracle Event Processing high availability deployment and notification groups. It simply requires that the new cluster member generate valid output events with respect to the point in time at which it begins processing input events.

A Type 2 application does not require a `warm-up-window-length` period.

Most applications will be Type 2 applications. It is common for an application to be brought up at an arbitrary point in time (on the primary Oracle Event Processing server), begin processing events from input streams at that point, and generate valid output events. In other words, the input stream is not paused while the application is started and input events are constantly being generated and arriving. It is reasonable to assume that in many cases a secondary node that does the same thing, but at a slightly different time, will also produce output events that are valid from the point of view of the application, although not necessarily identical to those events produced by the primary because of slight timing differences. For example, a financial application that only runs while the market is open might operate as a Type 2 application as follows: all servers can be brought up before the market opens and will begin processing incoming events at the same point in the market data stream. Multiple secondaries can be run to protect against failure and as long as the number of secondaries is sufficient while the market is open, there is no need to restart any secondaries that fail nor add additional secondaries, so no secondary needs to recover state.

**Ensure Applications are Idempotent** You should be able to run two copies of an application on different servers and they should not conflict in a shared cache or database. If you are using an external relation (such as a cache or table), then you must ensure that when a Oracle Event Processing server rejoins the cluster, your application is accessing the same cache or table as before: it must be joining against the same external relation again. The data source defined on the server must not have been changed; must ensure you're pulling data from same data source.

**Source Event Identity Externally** Many high availability solutions require that events be correlated between different servers and to do this events need to be universally identifiable. The best way to do this is use external information – preferably a

timestamp – to seed the event, rather than relying on the Oracle Event Processing system to provide this.

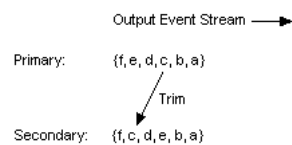
For more information, see [Section , "Prefer Application Time"](#).

**Understand the Importance of Event Ordering** For Oracle Event Processing high availability quality of service options that use queue trimming, not only must primary and secondary servers generate the same output events, but they must also generate them in exactly the same order.

Primary and secondary servers must generate the same output events and in exactly the same order when you choose Oracle Event Processing high availability quality of service options that use queue trimming and equality-based event identify (that is, nonmonotonic event identifiers - event identifiers that do not increase continually). In this case, generating output events in different orders can lead to either missed output events or unnecessary duplicate output events when there is a failure

Consider the output event streams shown in [Figure 24–8](#). The primary has output events a, b, and c. After outputting event c, the primary sends the secondary a queue trimming message.

**Figure 24–8 Event Order**



The secondary trims all events in its queue generated prior to event c including event c itself. In this case, the set of events trimmed will be {a, b, e, d, c} which is wrong because the primary has not yet output events d and e. If a failover occurs after processing the trimming message for event c, events will be lost.

To manage event ordering, consider the following design patterns:

- [Section , "Prefer Deterministic Behavior"](#)
- [Section , "Avoid Multithreading"](#)
- [Section , "Prefer Monotonic Event Identifiers"](#)

**Prefer Deterministic Behavior** In order for an application to generate events in the same order when run on multiple instances, it must be deterministic. The application must not rely on things like:

- Random number generator that may return different results on different machines.
- Methods like `System.currentTimeMillis` or `System.nanoTime` which can return different results on different machines because the system clocks are not synchronized.

**Avoid Multithreading** Because thread scheduling algorithms are very timing dependent, multithreading can be a source of nondeterministic behavior in applications. That is, different threads can be scheduled at different times on different machines.

For example, avoid creating an EPN in which multiple threads send events to an Oracle Event Processing high availability adapter in parallel. If such a channel is an event source for an Oracle Event Processing high availability adapter, it would cause events to be sent to the adapter in parallel by different threads and could make the event order nondeterministic.

For more information on channel configuration to avoid, see:

- [Section , "EventPartitioner"](#)
- `max-threads` in [Table C-9, " Attributes of the wlevs:channel Application Assembly Element"](#)

**Prefer Monotonic Event Identifiers** Event identifiers may be monotonic or nonmonotonic.

A monotonic identifier is one that increases continually (such as a time value).

A nonmonotonic identifier does not increase continually and may contain duplicates.

In general, you should design your Oracle Event Processing application using monotonic event identifiers. Using a monotonic event identifier, the Oracle Event Processing high availability adapter can handle an application that may produce events out of order.

**Write Oracle CQL Queries with High Availability in Mind** Not all Oracle CQL query usage is supported when using Oracle Event Processing high availability. You may need to redefine your Oracle CQL queries to address these restrictions.

For more information, see [Section , "Oracle CQL Query Restrictions"](#).

**Avoid Coupling Servers** The most performant high availability for Oracle Event Processing systems is when servers can run without requiring coordination between them. Generally this can be achieved if there is no shared state and the downstream system can tolerate duplicates. Increasing levels of high availability are targeted at increasing the fidelity of the stream of events that the downstream system sees, but this increasing fidelity comes with a performance penalty.

**Plan for Server Recovery** When a secondary server rejoins the multi-server domain, the server must have time to rebuild the Oracle Event Processing application state to match that of the current primary and active secondaries as [Section , "Choose an Adequate warm-up-window-length Time"](#) describes.

The time it takes for a secondary server to become available as an active secondary after rejoining the multi-server domain will be a factor in the number of active secondaries you require.

If a secondary is declared to be the new primary before it is ready, the secondary will throw an exception.

### Oracle CQL Query Restrictions

When writing Oracle CQL queries in an Oracle Event Processing application that uses Oracle Event Processing high availability options, observe the following restrictions:

- [Section , "Range-Based Windows"](#)
- [Section , "Tuple-Based Windows"](#)
- [Section , "Partitioned Windows"](#)
- [Section , "Sliding Windows"](#)
- [Section , "DURATION Clause and Non-Event Detection"](#)
- [Section , "Prefer Application Time"](#)

For more information on Oracle CQL, see the *Oracle Fusion Middleware CQL Language Reference for Oracle Event Processing*.

**Range-Based Windows** In a Type 1 application (where the application must generate exactly the same sequence of output events as existing secondaries), all range-based Oracle CQL windows must be shorter than the `warm-up-window-length` time. See also [Section , "Choose an Adequate warm-up-window-length Time"](#).

Channels must use application time if Oracle CQL queries contain range-based Windows. See also [Section , "Prefer Application Time"](#).

For more information, see "Range-Based Stream-to-Relation Window Operators" in the *Oracle Fusion Middleware CQL Language Reference for Oracle Event Processing*.

**Tuple-Based Windows** In a Type 1 application (where the application must generate exactly the same sequence of output events as existing secondaries), all tuple-based windows must also be qualified by time. See also [Section , "Choose an Adequate warm-up-window-length Time"](#).

For more information, see "Tuple-Based Stream-to-Relation Window Operators" in the *Oracle Fusion Middleware CQL Language Reference for Oracle Event Processing*.

**Partitioned Windows** Consider avoiding partitioned windows: there are cases where a partition cannot be rebuilt. If using partitioned windows, configure a `warm-up-window-length` time long enough to give the Oracle Event Processing server time to rebuild the partition. See also [Section , "Choose an Adequate warm-up-window-length Time"](#).

For more information, see "Partitioned Stream-to-Relation Window Operators" in the *Oracle Fusion Middleware CQL Language Reference for Oracle Event Processing*.

**Sliding Windows** Oracle CQL queries should not use sliding windows if new nodes that join the multi-server domain are expected to generate exactly the same output events as existing nodes.

For more information, see:

- [Section , "Rejoining the High Availability Multi-Server Domain"](#)
- "S[range T1 slide T2]" in the *Oracle Fusion Middleware CQL Language Reference for Oracle Event Processing*
- "S [rows N1 slide N2]" in the *Oracle Fusion Middleware CQL Language Reference for Oracle Event Processing*
- "S [partition by A1,..., Ak rows N range T1 slide T2]" in the *Oracle Fusion Middleware CQL Language Reference for Oracle Event Processing*

**DURATION Clause and Non-Event Detection** You must use application time if Oracle CQL queries contain a `DURATION` clause for non-event detection.

For more information, see:

- [Section , "Prefer Application Time"](#)
- "DURATION Clause" in the *Oracle Fusion Middleware CQL Language Reference for Oracle Event Processing*

**Prefer Application Time** In Oracle Event Processing each event is associated with a point in time at which the event occurred. Oracle CQL recognizes two types of time:

- Application time: a time value assigned to each event outside of Oracle CQL by the application before the event enters the Oracle CQL processor.

- System time: a time value associated with an event when it arrives at the Oracle CQL processor, essentially by calling `System.nanoTime()`.

Application time is generally the best approach for applications that need to be highly available. The application time is associated with an event before the event is sent to Oracle Event Processing, so it is consistent across active primary and secondary instances. System time, on the other hand, can cause application instances to generate different results since the time value associated with an event can be different on each instance due to system clocks not being synchronized.

You can use system time for applications whose Oracle CQL queries do not use time-based windows. Applications that use only event-based windows depend only on the arrival order of events rather than the arrival time, so you may use system time in this case.

If you must use system time with Oracle CQL queries that do use time-based windows, then you must use a special Oracle Event Processing high availability input adapter that intercepts incoming events and assigns a consistent time that spans primary and secondary instances.

## Configuring High Availability

This section describes how to configure high availability for your Oracle Event Processing application to provide the quality of service you require, including information on configuring failover, recovery and queue trimming, as well as configuring high availability adapters.

This section includes the following sections:

- [Section , "Configuring High Availability Quality of Service"](#)
- [Section , "Configuring High Availability Adapters"](#)

### Configuring High Availability Quality of Service

You configure Oracle Event Processing high availability quality of service in the EPN assembly file and component configuration files. For general information about these configuration files, see:

- [Section , "Overview of EPN Assembly Files"](#)
- [Section , "Overview of Component Configuration Files"](#)

---

---

**Note:** After making any Oracle Event Processing high availability configuration changes, you must redeploy your Oracle Event Processing application. See [Section , "Deploying Oracle Event Processing Applications"](#).

---

---

This section describes:

- [Section , "How to Configure Simple Failover"](#)
- [Section , "How to Configure Simple Failover With Buffering"](#)
- [Section , "How to Configure Light-Weight Queue Trimming"](#)
- [Section , "How to Configure Precise Recovery With JMS"](#)

For more information on configuring an Oracle Event Processing high availability application for scalability, see [Chapter 25, "Developing Scalable Applications"](#).

## How to Configure Simple Failover

You configure simple failover using the Oracle Event Processing buffering output adapter with a sliding window size of zero (0).

This procedure starts with the example EPN that [Figure 24–9](#) shows and adds the required components to configure it for simple failover. [Example 24–1](#) shows the corresponding EPN assembly file and [Example 24–2](#) shows the corresponding component configuration file.

For more information about this Oracle Event Processing high availability quality of service, see [Section , "Simple Failover"](#).

**Figure 24–9 Simple Failover EPN**



### Example 24–1 Simple Failover EPN Assembly File

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

  <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>
```

### Example 24–2 Simple Failover Component Configuration Assembly File

```
<?xml version="1.0" encoding="UTF-8"?>
<wlevs:config
  xmlns:wlevs="http://www.bea.com/ns/wlevs/config/application"
  xmlns:ha="http://www.oracle.com/ns/cep/config/cluster">
  <processor>
    <name>helloworldProcessor</name>
    <rules>
      <query id="helloworldRule">
        <![CDATA[ select * from helloworldInputChannel [Now] ]]>
      </query>
```

```

    </rules>
  </processor>
</wlevs:config>

```

### To configure simple failover:

1. Create a multi-server domain using Oracle Coherence.

For more information, see:

- "How to Create an Oracle Event Processing Multi-Server Domain With Default Groups Using Oracle Coherence" in the *Oracle Fusion Middleware Administrator's Guide for Oracle Event Processing*.
- "How to Create an Oracle Event Processing Multi-Server Domain With Custom Groups Using Oracle Coherence" in the *Oracle Fusion Middleware Administrator's Guide for Oracle Event Processing*.

2. Create an Oracle Event Processing application.

For more information, see [Section , "Creating Oracle Event Processing Projects"](#).

3. Edit the MANIFEST.MF file to add the following Import-Package entries:

- com.bea.wlevs.ede.api.cluster
- com.oracle.cep.cluster.hagroups
- com.oracle.cep.cluster.ha.adapter
- com.oracle.cep.cluster.ha.api

For more information, see [Section , "How to Add an OSGi Bundle to an Oracle Event Processing Project"](#).

4. Configure your Oracle Event Processing application EPN assembly file to add an Oracle Event Processing high availability buffering output adapter as [Example 24-3](#) shows.

- Add a wlevs:adapter element with provider set to ha-buffering after channel helloworldOutputChannel.
- Update the wlevs:listener element in channel helloworldOutputChannel to reference the ha-buffering adapter by its id.
- Add a wlevs:listener element to the ha-buffering adapter that references the HelloWorldBean class.

### Example 24-3 Simple Failover EPN Assembly File: Buffering Output Adapter

```

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

  <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:channel>

```

```

<wlevs:processor id="helloworldProcessor" />

<wlevs:channel id="helloworldOutputChannel" event-type="HelloWorldEvent"
advertise="true">
  <wlevs:listener ref="myHaSlidingWindowAdapter"/>
  <wlevs:source ref="helloworldProcessor"/>
</wlevs:channel>

<wlevs:adapter id="myHaSlidingWindowAdapter" provider="ha-buffering" >
  <wlevs:listener>
    <bean class="com.bea.wlevs.example.helloworld.HelloWorldBean"/>
  </wlevs:listener>
</wlevs:adapter>

</beans>

```

5. Optionally, configure the channel downstream from the input adapter (helloworldInputChannel) to configure an application timestamp based on an appropriate event property as [Example 24-4](#) shows.

For simple failover, you can use system timestamps because events are not correlated between servers. However, it is possible that slightly different results might be output from the buffer if application timestamps are not used.

In this example, event property arrivalTime is used.

The wlevs:expression should be set to this event property.

#### **Example 24-4 Application Timestamp Configuration**

```

...
<wlevs:channel id="helloworldInputChannel" event-type="HelloWorldEvent" >
  <wlevs:listener ref="helloworldProcessor"/>
  <wlevs:source ref="myHaInputAdapter"/>
  <wlevs:application-timestamped>
    <wlevs:expression>arrivalTime</wlevs:expression>
  </wlevs:application-timestamped>
</wlevs:channel>
...

```

6. Configure the Oracle Event Processing high availability buffering output adapter. Set the instance property windowLength to zero (0) as [Example 24-5](#) shows.

#### **Example 24-5 Configuring windowLength in the Buffering Output Adapter**

```

...
<wlevs:adapter id="myHaSlidingWindowAdapter" provider="ha-buffering" >
  <wlevs:listener>
    <bean class="com.bea.wlevs.example.helloworld.HelloWorldBean"/>
  </wlevs:listener>
  <wlevs:instance-property name="windowLength" value="0"/>
</wlevs:adapter>
...

```

For more information, see [Section , "Buffering Output Adapter EPN Assembly File Configuration"](#).

7. Optionally, configure the component configuration file to include the Oracle Event Processing high availability buffering output adapter as [Example 24-6](#) shows.



### Example 24–6 Simple Failover Component Configuration File With High Availability Adapters

```
<?xml version="1.0" encoding="UTF-8"?>
<wlevs:config
  xmlns:wlevs="http://www.bea.com/ns/wlevs/config/application"
  xmlns:ha="http://www.oracle.com/ns/cep/config/cluster">
  <processor>
    <name>helloworldProcessor</name>
    <rules>
      <query id="helloworldRule">
        <![CDATA[ select * from helloworldInputChannel [Now] ]]>
      </query>
    </rules>
  </processor>

  <ha:ha-buffering-adapter >
    <name>myHaSlidingWindowAdapter</name>
    <window-length>0</window-length>
  </ha:ha-buffering-adapter >

</wlevs:config>
```

For more information, see:

- [Section , "Buffering Output Adapter Component Configuration File Configuration"](#)
8. Deploy your application to the deployment group you created in step 1.

For more information, see [Section , "Deploying Oracle Event Processing Applications"](#).

Oracle Event Processing automatically selects one of the Oracle Event Processing servers as the primary.

### How to Configure Simple Failover With Buffering

You configure simple failover using the Oracle Event Processing buffering output adapter with a sliding window size greater than zero (0).

This procedure starts with the example EPN that [Figure 24–10](#) shows and adds the required components to configure it for simple failover with buffering. [Example 24–7](#) shows the corresponding EPN assembly file and [Example 24–8](#) shows the corresponding component configuration file.

For more information about this Oracle Event Processing high availability quality of service, see [Section , "Simple Failover with Buffering"](#).

**Figure 24–10 Simple Failover With Buffering EPN**



### Example 24–7 Simple Failover With Buffering EPN Assembly File

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

  <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: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>

```

#### **Example 24–8 Simple Failover With Buffering Component Configuration Assembly File**

```

<?xml version="1.0" encoding="UTF-8"?>
<wlevs:config
  xmlns:wlevs="http://www.bea.com/ns/wlevs/config/application"
  xmlns:ha="http://www.oracle.com/ns/cep/config/cluster">
  <processor>
    <name>helloworldProcessor</name>
    <rules>
      <query id="helloworldRule">
        <![CDATA[ select * from helloworldInputChannel [Now] ]]>
      </query>
    </rules>
  </processor>
</wlevs:config>

```

#### **To configure simple failover with buffering:**

1. Create a multi-server domain using Oracle Coherence.

For more information, see:

- "How to Create an Oracle Event Processing Multi-Server Domain With Default Groups Using Oracle Coherence" in the *Oracle Fusion Middleware Administrator's Guide for Oracle Event Processing*.
- "How to Create an Oracle Event Processing Multi-Server Domain With Custom Groups Using Oracle Coherence" in the *Oracle Fusion Middleware Administrator's Guide for Oracle Event Processing*.

2. Create an Oracle Event Processing application.

For more information, see [Section , "Creating Oracle Event Processing Projects"](#).

3. Edit the MANIFEST.MF file to add the following `Import-Package` entries:

- `com.bea.wlevs.ede.api.cluster`
- `com.oracle.cep.cluster.hagroups`
- `com.oracle.cep.cluster.ha.adapter`
- `com.oracle.cep.cluster.ha.api`

For more information, see [Section , "How to Add an OSGi Bundle to an Oracle Event Processing Project"](#).

4. Configure your Oracle Event Processing application EPN assembly file to add an Oracle Event Processing high availability buffering output adapter as [Example 24–3](#) shows.
  - Add a `wlevs:adapter` element with `provider` set to `ha-buffering` after channel `helloworldOutputChannel`.
  - Update the `wlevs:listener` element in channel `helloworldOutputChannel` to reference the `ha-buffering` adapter by its `id`.
  - Add a `wlevs:listener` element to the `ha-buffering` adapter that references the `HelloWorldBean` class.

**Example 24–9 Simple Failover EPN Assembly File: Buffering Output Adapter**

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

  <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:channel>

  <wlevs:processor id="helloworldProcessor" />

  <wlevs:channel id="helloworldOutputChannel" event-type="HelloWorldEvent"
    advertise="true">
    <wlevs:listener ref="myHaSlidingWindowAdapter"/>
    <wlevs:source ref="helloworldProcessor"/>
  </wlevs:channel>

  <wlevs:adapter id="myHaSlidingWindowAdapter" provider="ha-buffering" >
    <wlevs:listener>
      <bean class="com.bea.wlevs.example.helloworld.HelloWorldBean"/>
    </wlevs:listener>
  </wlevs:adapter>

</beans>
```

5. Optionally, configure the channel downstream from the input adapter (`helloworldInputChannel`) to configure an application timestamp based on an appropriate event property as [Example 24–10](#) shows.

For simple failover with buffering, you can use system timestamps because events are not correlated between servers. However, it is possible that slightly different results might be output from the buffer if application timestamps are not used.

In this example, event property `arrivalTime` is used.

The `wlevs:expression` should be set to this event property.

**Example 24–10 Application Timestamp Configuration**

```

...
<wlevs:channel id="helloworldInputChannel" event-type="HelloWorldEvent" >
  <wlevs:listener ref="helloworldProcessor"/>
  <wlevs:source ref="myHaInputAdapter"/>
  <wlevs:application-timestamped>
    <wlevs:expression>arrivalTime</wlevs:expression>
  </wlevs:application-timestamped>
</wlevs:channel>
...

```

6. Configure the Oracle Event Processing high availability buffering output adapter. Set the instance property `windowLength` to a value greater than zero (0) as [Example 24–11](#) shows.

**Example 24–11 Configuring windowLength in the Buffering Output Adapter**

```

...
<wlevs:adapter id="myHaSlidingWindowAdapter" provider="ha-buffering" >
  <wlevs:listener>
    <bean class="com.bea.wlevs.example.helloworld.HelloWorldBean"/>
  </wlevs:listener>
  <wlevs:instance-property name="windowLength" value="15000"/>
</wlevs:adapter>
...

```

For more information, see [Section , "Buffering Output Adapter EPN Assembly File Configuration"](#).

7. Optionally, configure the component configuration file to include the Oracle Event Processing high availability buffering output adapter as [Example 24–12](#) shows.

**Example 24–12 Simple Failover With Buffering Component Configuration File**

```

<?xml version="1.0" encoding="UTF-8"?>
<wlevs:config
  xmlns:wlevs="http://www.bea.com/ns/wlevs/config/application"
  xmlns:ha="http://www.oracle.com/ns/cep/config/cluster">
  <processor>
    <name>helloworldProcessor</name>
    <rules>
      <query id="helloworldRule">
        <![CDATA[ select * from helloworldInputChannel [Now] ]]>
      </query>
    </rules>
  </processor>

  <ha:ha-buffering-adapter >
    <name>myHaSlidingWindowAdapter</name>
    <window-length>15000</window-length>
  </ha:ha-buffering-adapter >
</wlevs:config>

```

For more information, see:

- [Section , "Buffering Output Adapter Component Configuration File Configuration"](#)
8. If your application is an Oracle Event Processing high availability Type 1 application (the application must generate exactly the same sequence of output events as existing secondaries), configure the `warm-up-window-length` for the buffering output adapter.

For more information, see:

- [Section , "Choose an Adequate warm-up-window-length Time"](#)
- [Section , "Buffering Output Adapter Component Configuration File Configuration"](#)

9. Deploy your application to the deployment group you created in step 1.

For more information, see [Section , "Deploying Oracle Event Processing Applications"](#).

Oracle Event Processing automatically selects one of the Oracle Event Processing servers as the primary.

## How to Configure Light-Weight Queue Trimming

You configure light-weight queue trimming using the Oracle Event Processing high availability input adapter and the broadcast output adapter.

This procedure starts with the example EPN that [Figure 24–11](#) shows and adds the required components to configure it for light-weight queue trimming. [Example 24–13](#) shows the corresponding EPN assembly file and [Example 24–14](#) shows the corresponding component configuration file.

For more information about this Oracle Event Processing high availability quality of service, see [Section , "Light-Weight Queue Trimming"](#).

**Figure 24–11 Light-Weight Queue Trimming EPN**



**Example 24–13 Light-Weight Queue Trimming EPN Assembly File**

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

  <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>
```

**Example 24–14 Light-Weight Queue Trimming Component Configuration Assembly File**

```
<?xml version="1.0" encoding="UTF-8"?>
<wlevs:config
  xmlns:wlevs="http://www.bea.com/ns/wlevs/config/application"
  xmlns:ha="http://www.oracle.com/ns/cep/config/cluster">
  <processor>
    <name>helloworldProcessor</name>
    <rules>
      <query id="helloworldRule">
        <![CDATA[ select * from helloworldInputChannel [Now] ]]>
      </query>
    </rules>
  </processor>
</wlevs:config>
```

**To configure light-weight queue trimming:**

1. Create a multi-server domain using Oracle Coherence.

For more information, see:

- "How to Create an Oracle Event Processing Multi-Server Domain With Default Groups Using Oracle Coherence" in the *Oracle Fusion Middleware Administrator's Guide for Oracle Event Processing*.
- "How to Create an Oracle Event Processing Multi-Server Domain With Custom Groups Using Oracle Coherence" in the *Oracle Fusion Middleware Administrator's Guide for Oracle Event Processing*.

2. Create an Oracle Event Processing application.

For more information, see [Section , "Creating Oracle Event Processing Projects"](#).

3. Edit the MANIFEST.MF file to add the following Import-Package entries:

- com.bea.wlevs.ede.api.cluster
- com.oracle.cep.cluster.hagroups
- com.oracle.cep.cluster.ha.adapter
- com.oracle.cep.cluster.ha.api

For more information, see [Section , "How to Add an OSGi Bundle to an Oracle Event Processing Project"](#).

4. Configure your Oracle Event Processing application EPN assembly file to add an Oracle Event Processing high availability input adapter as [Example 24–15](#) shows:

- Add a wlevs:adapter element with provider set to ha-inbound after the regular input adapter helloworldAdapter.
- Add a wlevs:listener element to the regular input adapter helloworldAdapter that references the ha-inbound adapter by its id.
- Add a wlevs:source element to the helloworldInputChannel that references the ha-inbound adapter by its id.

**Example 24–15 Light-Weight Queue Trimming EPN Assembly File: High Availability Input Adapter**

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

```

<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:listener ref="myHaInputAdapter"/>
</wlevs:adapter>

<wlevs:adapter id="myHaInputAdapter" provider="ha-inbound" >
</wlevs:adapter>

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

```

5. Configure your Oracle Event Processing application EPN assembly file to add an Oracle Event Processing high availability broadcast output adapter as [Example 24–16](#) shows.

- Add a `wlevs:adapter` element with provider set to `ha-broadcast` after channel `helloworldOutputChannel`.
- Update the `wlevs:listener` element in channel `helloworldOutputChannel` to reference the `ha-broadcast` adapter by its id.
- Add a `wlevs:listener` element to the `ha-broadcast` adapter that references the `HelloWorldBean` class.

**Example 24–16 Light-Weight Queue Trimming EPN Assembly File: Broadcast Output Adapter**

```

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

  <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:listener ref="myHaInputAdapter"/>
  </wlevs:adapter>

  <wlevs:adapter id="myHaInputAdapter" provider="ha-inbound" >
  </wlevs:adapter>

```

```

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

<wlevs:processor id="helloworldProcessor" />

<wlevs:channel id="helloworldOutputChannel" event-type="HelloWorldEvent"
  advertise="true">
  <wlevs:listener ref="myHaBroadcastAdapter"/>
  <wlevs:source ref="helloworldProcessor"/>
</wlevs:channel>

<wlevs:adapter id="myHaBroadcastAdapter" provider="ha-broadcast" >
  <wlevs:listener>
    <bean class="com.bea.wlevs.example.helloworld.HelloWorldBean"/>
  </wlevs:listener>
</wlevs:adapter>

</beans>

```

## 6. Configure the Oracle Event Processing high availability input adapter.

Consider the following example configurations:

- [Example 24–17, "High Availability Input Adapter: Default Configuration"](#)
- [Example 24–18, "High Availability Input Adapter: Tuple Events"](#)
- [Example 24–19, "High Availability Input Adapter: Key of One Event Property"](#)
- [Example 24–20, "High Availability Input Adapter: Key of Multiple Event Properties"](#)

For more information, see [Section , "High Availability Input Adapter EPN Assembly File Configuration"](#).

### **Example 24–17 High Availability Input Adapter: Default Configuration**

This example shows a high availability input adapter configuration using all defaults. The mandatory key is based on all event properties and the event property that the high availability input adapter assigns a time value to is an event property named `arrivalTime`.

```

...
<wlevs:adapter id="myHaInputAdapter" provider="ha-inbound" >
  <wlevs:instance-property name="timeProperty" value="arrivalTime"/>
</wlevs:adapter>
...

```

### **Example 24–18 High Availability Input Adapter: Tuple Events**

This example shows a high availability input adapter configuration using all defaults. The mandatory key is based on all event properties and the event property that the high availability input adapter assigns a time value to is an event property named `arrivalTime`. Because the events are tuple-based events, you must specify the event type (`MyEventType`) using the `eventType` property.

```

...
<wlevs:adapter id="myHaInputAdapter" provider="ha-inbound" >
  <wlevs:instance-property name="timeProperty" value="arrivalTime"/>
  <wlevs:instance-property name="eventType" value="MyEventType"/>
</wlevs:adapter>
...

```



**Example 24–19 High Availability Input Adapter: Key of One Event Property**

This example shows a high availability input adapter configuration where the mandatory key is based on one event property (named `id`) and the event property that the high availability input adapter assigns a time value to is an event property named `arrivalTime`.

```
...
<wlevs:adapter id="myHaInputAdapter" provider="ha-inbound" >
  <wlevs:instance-property name="keyProperties" value="id"/>
  <wlevs:instance-property name="timeProperty" value="arrivalTime"/>
</wlevs:adapter>
...
```

**Example 24–20 High Availability Input Adapter: Key of Multiple Event Properties**

This example shows a high availability input adapter configuration where the mandatory key is based on more than one event property (properties `orderId` and `accountId`) and the event property that the high availability input adapter assigns a time value to is an event property named `arrivalTime`. A compound key Java class (`com.acme.MyCompoundKeyClass`) is mandatory and its implementation is shown in [Example 24–21](#). The `hashCode` and `equals` methods are required. When you specify a `keyClass`, the `keyProperties` instance property is ignored: Oracle Event Processing assumes that the compound key is based on all the getter methods in the `keyClass`.

```
...
<wlevs:adapter id="myHaInputAdapter" provider="ha-inbound" >
  <wlevs:instance-property name="timeProperty" value="arrivalTime"/>
  <wlevs:instance-property name="keyClass" value="com.acme.MyCompoundKeyClass"/>
</wlevs:adapter>
...
```

**Example 24–21 MyCompoundKeyClass Implementation**

```
package com.acme;

public class MyCompoundKeyClass {
    private int orderId;
    private int accountId;

    public MyCompoundKeyClass() {}

    public int getOrderId() {
        return orderId;
    }
    public setOrderId(int orderId) {
        this.orderId = orderId;
    }
    public int getAccountId() {
        return accountId;
    }
    public setOrderId(int accountId) {
        this.accountId = accountId;
    }

    public int hashCode() {
        int hash = 1;
        hash = hash * 31 + orderId.hashCode();
        hash = hash * 31 + (accountId == null ? 0 : accountId.hashCode());
        return hash;
    }

    public boolean equals(Object obj) {
        if (obj == this) return true;
        if (obj == null) return false;
```

```

        if (!(obj instanceof MyCompoundKeyClass)) return false;
        MyCompoundKeyClass k = (MyCompoundKeyClass) obj;
        return k.accountID == accountID && k.orderID == orderID;
    }
}

```

7. Configure the channel downstream from the high availability input adapter (helloworldInputChannel) to configure an application timestamp based on the high availability input adapter timeProperty setting as [Example 24–22](#) shows.

The `wlevs:expression` should be set to the `timeProperty` value.

#### **Example 24–22 Application Timestamp Configuration**

```

...
<wlevs:adapter id="myHaInputAdapter" provider="ha-inbound" >
  <wlevs:instance-property name="keyProperties" value="id"/>
  <wlevs:instance-property name="eventType" value="HelloWorldEvent"/>
  <wlevs:instance-property name="timeProperty" value="arrivalTime"/>
</wlevs:adapter>

<wlevs:channel id="helloworldInputChannel" event-type="HelloWorldEvent" >
  <wlevs:listener ref="helloworldProcessor"/>
  <wlevs:source ref="myHaInputAdapter"/>
  <wlevs:application-timestamped>
    <wlevs:expression>arrivalTime</wlevs:expression>
  </wlevs:application-timestamped>
</wlevs:channel>
...

```

8. Configure the Oracle Event Processing high availability broadcast output adapter.

Consider the following example configurations:

- [Example 24–23, "Broadcast Output Adapter: Default Configuration"](#)
- [Example 24–24, "Broadcast Output Adapter: Key of One Event Property"](#)
- [Example 24–25, "Broadcast Output Adapter: Key of Multiple Event Properties"](#)

For more information, see [Section , "Broadcast Output Adapter EPN Assembly File Configuration"](#).

#### **Example 24–23 Broadcast Output Adapter: Default Configuration**

This example shows a broadcast output adapter configuration using all defaults. The mandatory key is based on all event properties, key values are nonmonotonic (do not increase continually) and total order (unique).

```

...
<wlevs:adapter id="myHaSlidingWindowAdapter" provider="ha-broadcast" >
  <wlevs:listener>
    <bean class="com.bea.wlevs.example.helloworld.HelloWorldBean"/>
  </wlevs:listener>
</wlevs:adapter>
...

```

#### **Example 24–24 Broadcast Output Adapter: Key of One Event Property**

This example shows a broadcast output adapter configuration where the mandatory key is based on one event property (named `timeProperty`), key values are monotonic (they do increase continually) and not total order (not unique).

```

...
<wlevs:adapter id="myHaSlidingWindowAdapter" provider="ha-broadcast" >
  <wlevs:listener>

```

```

        <bean class="com.bea.wlevs.example.helloworld.HelloWorldBean"/>
    </wlevs:listener>
    <wlevs:instance-property name="keyProperties" value="timeProperty"/>
    <wlevs:instance-property name="monotonic" value="true"/>
    <wlevs:instance-property name="totalOrder" value="false"/>
</wlevs:adapter>
...

```

#### **Example 24–25 Broadcast Output Adapter: Key of Multiple Event Properties**

This example shows a broadcast output adapter configuration where the mandatory key is based on more than one event property (properties `timeProperty` and `accountID`), key values are monotonic (they do increase continually) and total order (unique). A compound key Java class (`com.acme.MyCompoundKeyClass`) is mandatory and its implementation is shown in [Example 24–26](#). The `hashCode` and `equals` methods are required. When you specify a `keyClass`, the `keyProperties` instance property is ignored: Oracle Event Processing assumes that the compound key is based on all the getter methods in the `keyClass`.

```

...
<wlevs:adapter id="myHaSlidingWindowAdapter" provider="ha-broadcast" >
    <wlevs:listener>
        <bean class="com.bea.wlevs.example.helloworld.HelloWorldBean"/>
    </wlevs:listener>
    <wlevs:instance-property name="keyClass" value="com.acme.MyCompoundKeyClass"/>
    <wlevs:instance-property name="monotonic" value="true"/>
    <wlevs:instance-property name="totalOrder" value="true"/>
</wlevs:adapter>
...

```

#### **Example 24–26 MyCompoundKeyClass Implementation**

```

package com.acme;

public class MyCompoundKeyClass {
    private int timeProperty;
    private int accountID;

    public MyCompoundKeyClass() {}

    public int getTimeProperty() {
        return orderID;
    }
    public setTimeProperty(int timeProperty) {
        this.timeProperty = timeProperty;
    }
    public int getAccountID() {
        return accountID;
    }
    public setOrderID(int accountID) {
        this.accountID = accountID;
    }

    public int hashCode() {
        int hash = 1;
        hash = hash * 31 + timeProperty.hashCode();
        hash = hash * 31 + (accountID == null ? 0 : accountID.hashCode());
        return hash;
    }

    public boolean equals(Object obj) {
        if (obj == this) return true;
        if (obj == null) return false;
        if (!(obj instanceof MyCompoundKeyClass)) return false;
        MyCompoundKeyClass k = (MyCompoundKeyClass) obj;

```

```

        return k.accountID == accountID && k.orderID == orderID;
    }
}

```

9. Optionally, configure the component configuration file to include the Oracle Event Processing high availability input adapter and buffering output adapter as [Example 24–27](#) shows.

**Example 24–27 Light-Weight Queue Trimming Component Configuration File**

```

<?xml version="1.0" encoding="UTF-8"?>
<wlevs:config
  xmlns:wlevs="http://www.bea.com/ns/wlevs/config/application"
  xmlns:ha="http://www.oracle.com/ns/cep/config/cluster">
  <processor>
    <name>helloworldProcessor</name>
    <rules>
      <query id="helloworldRule">
        <![CDATA[ select * from helloworldInputChannel [Now] ]]>
      </query>
    </rules>
  </processor>

  <ha:ha-inbound-adapter>
    <name>myHaInputAdapter</name>
  </ha:ha-inbound-adapter>

  <ha:ha-broadcast-adapter>
    <name>myHaBroadcastAdapter</name>
    <trimming-interval units="events">10</trimming-interval>
  </ha:ha-broadcast-adapter>
</wlevs:config>

```

For more information, see:

- [Section , "High Availability Input Adapter Component Configuration File Configuration"](#)
  - [Section , "Broadcast Output Adapter Component Configuration File Configuration"](#)
10. If your application is an Oracle Event Processing high availability Type 1 application (the application must generate exactly the same sequence of output events as existing secondaries), configure the `warm-up-window-length` for the broadcast output adapter.

For more information, see:

- [Section , "Choose an Adequate warm-up-window-length Time"](#)
  - [Section , "Broadcast Output Adapter Component Configuration File Configuration"](#)
11. Deploy your application to the deployment group you created in step 1.
- For more information, see [Section , "Deploying Oracle Event Processing Applications"](#).

Oracle Event Processing automatically selects one of the Oracle Event Processing servers as the primary.

## How to Configure Precise Recovery With JMS

You configure precise recovery with JMS using the Oracle Event Processing high availability input adapter and correlating output adapter.

This procedure describes how to create the example EPN that [Figure 24–12](#) shows. [Example 24–28](#) shows the corresponding EPN assembly file and [Example 24–29](#) shows the corresponding component configuration file.

For more information about this Oracle Event Processing high availability quality of service, see [Section , "Precise Recovery with JMS"](#).

---



---

**Note:** The JMS destination used by JMS adapters for precise recovery must be topics, rather than queues.

---



---

**Figure 24–12** *Precise Recovery With JMS EPN*



### Example 24–28 *Precise Recovery With JMS EPN Assembly File*

```

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

  <wlevs:event-type-repository>
    <wlevs:event-type type-name="StockTick">
      <wlevs:properties>
        <wlevs:property name="lastPrice" type="double" />
        <wlevs:property name="symbol" type="char" />
      </wlevs:properties>
    </wlevs:event-type>
  </wlevs:event-type-repository>

  <wlevs:adapter id="JMSInboundAdapter" provider="jms-inbound">
    <wlevs:listener ref="myHaInputAdapter"/>
  </wlevs:adapter>

  <wlevs:adapter id="myHaInputAdapter" provider="ha-inbound" >
    <wlevs:instance-property name="keyProperties" value="sequenceNo"/>
    <wlevs:instance-property name="timeProperty" value="inboundTime"/>
  </wlevs:adapter>

  <wlevs:channel id="channel1" event-type="StockTick">
    <wlevs:listener ref="processor1" />
    <wlevs:source ref="myHaInputAdapter"/>
    <wlevs:application-timestamped>
      <wlevs:expression>inboundTime</wlevs:expression>
    </wlevs:application-timestamped>
  </wlevs:channel>

  <wlevs:processor id="processor1">
    <wlevs:listener ref="channel2" />
  </wlevs:processor>

  <wlevs:adapter id="myHaCorrelatingAdapter" provider="ha-correlating" >
    <wlevs:instance-property name="correlatedSource" ref="clusterCorrelatingOutstream"/>
    <wlevs:instance-property name="failOverDelay" value="2000"/>
    <wlevs:listener ref="JMSOutboundAdapter"/>
  </wlevs:adapter>

```

```

</wlevs:adapter>

<wlevs:channel id="channel2" event-type="StockTick">
  <wlevs:listener ref="myHaCorrelatingAdapter" />
</wlevs:channel>

<wlevs:adapter id="JMSOutboundAdapter" provider="jms-outbound">
</wlevs:adapter>

<wlevs:adapter id="JMSInboundAdapter2" provider="jms-inbound">
</wlevs:adapter>

<wlevs:channel id="clusterCorrelatingOutstream" event-type="StockTick" advertise="true">
  <wlevs:source ref="JMSInboundAdapter2"/>
</wlevs:channel>
</beans>

```

### Example 24–29 Precise Recovery With JMS Component Configuration Assembly File

```

<?xml version="1.0" encoding="UTF-8"?>
<wlevs:config
  xmlns:wlevs="http://www.bea.com/ns/wlevs/config/application"
  xmlns:ha="http://www.oracle.com/ns/cep/config/cluster">
  <processor>
    <name>processor1</name>
    <rules>
      <query id="helloworldRule">
        <![CDATA[ select * from channel1 [Now] ]]>
      </query>
    </rules>
  </processor>
</wlevs:config>

```

#### To configure precise recovery with JMS:

1. Create a multi-server domain using Oracle Coherence.

For more information, see:

- "How to Create an Oracle Event Processing Multi-Server Domain With Default Groups Using Oracle Coherence" in the *Oracle Fusion Middleware Administrator's Guide for Oracle Event Processing*.
- "How to Create an Oracle Event Processing Multi-Server Domain With Custom Groups Using Oracle Coherence" in the *Oracle Fusion Middleware Administrator's Guide for Oracle Event Processing*.

2. Create an Oracle Event Processing application.

For more information, see [Section , "Creating Oracle Event Processing Projects"](#).

3. Edit the MANIFEST.MF file to add the following `Import-Package` entries:

- `com.bea.wlevs.ede.api.cluster`
- `com.oracle.cep.cluster.hagroups`
- `com.oracle.cep.cluster.ha.adapter`
- `com.oracle.cep.cluster.ha.api`

For more information, see [Section , "How to Add an OSGi Bundle to an Oracle Event Processing Project"](#).

4. Configure your Oracle Event Processing application EPN assembly file to add an Oracle Event Processing high availability input adapter as [Example 24–30](#) shows:

- Add a `wlevs:adapter` element with provider set to `ha-inbound` after the regular input adapter `JMSInboundAdapter`.
- Add a `wlevs:listener` element to the regular input adapter `JMSInboundAdapter` that references the `ha-inbound` adapter by its id.
- Add a `wlevs:source` element to the channel `channel1` that references the `ha-inbound` adapter by its id.

**Example 24–30 Precise Recovery With JMS EPN Assembly File: High Availability Input Adapter**

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

  <wlevs:event-type-repository>
    <wlevs:event-type type-name="StockTick">
      <wlevs:properties>
        <wlevs:property name="lastPrice" type="double" />
        <wlevs:property name="symbol" type="char" />
      </wlevs:properties>
    </wlevs:event-type>
  </wlevs:event-type-repository>

  <wlevs:adapter id="JMSInboundAdapter" provider="jms-inbound">
    <wlevs:listener ref="myHaInputAdapter"/>
  </wlevs:adapter>

  <wlevs:adapter id="myHaInputAdapter" provider="ha-inbound" >
  </wlevs:adapter>

  <wlevs:channel id="channel1" event-type="StockTick">
    <wlevs:listener ref="processor1" />
    <wlevs:source ref="myHaInputAdapter"/>
  </wlevs:channel>

  ...

</beans>
```

5. Configure your Oracle Event Processing application EPN assembly file to add an Oracle Event Processing high availability correlating output adapter as [Example 24–31](#) shows.

- Add a `wlevs:adapter` element with provider set to `ha-correlating` after channel `channel2`.
- Update the `wlevs:listener` element in channel `channel2` to reference the `ha-correlating` adapter by its id.
- Add a `wlevs:listener` element to the `ha-correlating` adapter that references the regular output adapter `JMSOutboundAdapter`.

**Example 24–31 Precise Recovery With JMS EPN Assembly File: Correlating Output Adapter**

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

  <wlevs:event-type-repository>
    <wlevs:event-type type-name="StockTick">
      <wlevs:properties>
        <wlevs:property name="lastPrice" type="double" />
        <wlevs:property name="symbol" type="char" />
      </wlevs:properties>
    </wlevs:event-type>
  </wlevs:event-type-repository>
```

```

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

    <wlevs:adapter id="JMSInboundAdapter" provider="jms-inbound">
        <wlevs:listener ref="myHaInputAdapter" />
    </wlevs:adapter>

    <wlevs:adapter id="myHaInputAdapter" provider="ha-inbound" >
    </wlevs:adapter>

    <wlevs:channel id="channel1" event-type="StockTick">
        <wlevs:listener ref="processor1" />
        <wlevs:source ref="myHaInputAdapter" />
    </wlevs:channel>

    <wlevs:processor id="processor1">
        <wlevs:listener ref="channel2" />
    </wlevs:processor>

    <wlevs:channel id="channel2" event-type="StockTick">
        <wlevs:listener ref="myHaCorrelatingAdapter" />
    </wlevs:channel>

    <wlevs:adapter id="myHaCorrelatingAdapter" provider="ha-correlating" >
        <wlevs:listener ref="JMSOutboundAdapter" />
    </wlevs:adapter>

    <wlevs:adapter id="JMSOutboundAdapter" provider="jms-outbound">
    </wlevs:adapter>

    ...
</beans>

```

## 6. Configure the Oracle Event Processing high availability input adapter.

Consider the following example configurations:

- [Example 24–32, "High Availability Input Adapter: Default Configuration"](#)
- [Example 24–33, "High Availability Input Adapter: Tuple Events"](#)
- [Example 24–34, "High Availability Input Adapter: Key of One Event Property"](#)
- [Example 24–35, "High Availability Input Adapter: Key of Multiple Event Properties"](#)

For more information, see [Section , "High Availability Input Adapter EPN Assembly File Configuration"](#).

### **Example 24–32 High Availability Input Adapter: Default Configuration**

This example shows a high availability input adapter configuration using all defaults. The mandatory key is based on all event properties and the event property that the high availability input adapter assigns a time value to is an event property named `arrivalTime`.

```

...
<wlevs:adapter id="myHaInputAdapter" provider="ha-inbound" >
    <wlevs:instance-property name="timeProperty" value="arrivalTime"/>
</wlevs:adapter>
...

```



**Example 24–33 High Availability Input Adapter: Tuple Events**

This example shows a high availability input adapter configuration using all defaults. The mandatory key is based on all event properties and the event property that the high availability input adapter assigns a time value to is an event property named `arrivalTime`. Because the events are tuple-based events, you must specify the event type (`MyEventType`) using the `eventType` property.

```
...
<wlevs:adapter id="myHaInputAdapter" provider="ha-inbound" >
  <wlevs:instance-property name="timeProperty" value="arrivalTime"/>
  <wlevs:instance-property name="eventType" value="MyEventType"/>
</wlevs:adapter>
...
```

**Example 24–34 High Availability Input Adapter: Key of One Event Property**

This example shows a high availability input adapter configuration where the mandatory key is based on one event property (named `sequenceNo`) and the event property that the high availability input adapter assigns a time value to is an event property named `inboundTime`.

```
...
<wlevs:adapter id="myHaInputAdapter" provider="ha-inbound" >
  <wlevs:instance-property name="keyProperties" value="sequenceNo"/>
  <wlevs:instance-property name="timeProperty" value="inboundTime"/>
</wlevs:adapter>
...
```

**Example 24–35 High Availability Input Adapter: Key of Multiple Event Properties**

This example shows a high availability input adapter configuration where the mandatory key is based on more than one event property (properties `orderId` and `accountId`) and the event property that the high availability input adapter assigns a time value to is an event property named `arrivalTime`. A compound key Java class (`com.acme.MyCompoundKeyClass`) is mandatory and its implementation is shown in [Example 24–36](#). The `hashCode` and `equals` methods are required. When you specify a `keyClass`, the `keyProperties` instance property is ignored: Oracle Event Processing assumes that the compound key is based on all the getter methods in the `keyClass`.

```
...
<wlevs:adapter id="myHaInputAdapter" provider="ha-inbound" >
  <wlevs:instance-property name="timeProperty" value="arrivalTime"/>
  <wlevs:instance-property name="keyClass" value="com.acme.MyCompoundKeyClass"/>
</wlevs:adapter>
...
```

**Example 24–36 MyCompoundKeyClass Implementation**

```
package com.acme;

public class MyCompoundKeyClass {
    private int orderId;
    private int accountId;

    public MyCompoundKeyClass() {}

    public int getOrderId() {
        return orderId;
    }
    public setOrderId(int orderId) {
        this.orderId = orderId;
    }
    public int getAccountId() {
```

```

        return accountID;
    }
    public setOrderID(int accountID) {
        this.accountID = accountID;
    }

    public int hashCode() {
        int hash = 1;
        hash = hash * 31 + orderID.hashCode();
        hash = hash * 31 + (accountID == null ? 0 : accountID.hashCode());
        return hash;
    }

    public boolean equals(Object obj) {
        if (obj == this) return true;
        if (obj == null) return false;
        if (!(obj instanceof MyCompoundKeyClass)) return false;
        MyCompoundKeyClass k = (MyCompoundKeyClass) obj;
        return k.accountID == accountID && k.orderID == orderID;
    }
}

```

7. Configure the channel downstream from the high availability input adapter (channel1) to configure an application timestamp based on the high availability input adapter timeProperty setting as [Example 24–37](#) shows.

The wlevs:expression should be set to the timeProperty value.

#### **Example 24–37 Application Timestamp Configuration**

```

...
<wlevs:adapter id="myHaInputAdapter" provider="ha-inbound" >
  <wlevs:instance-property name="eventType" value="HelloWorldEvent"/>
  <wlevs:instance-property name="keyProperties" value="sequenceNo"/>
  <wlevs:instance-property name="timeProperty" value="inboundTime"/>
</wlevs:adapter>

<wlevs:channel id="channel1" event-type="StockTick">
  <wlevs:listener ref="processor1" />
  <wlevs:source ref="myHaInputAdapter"/>
  <wlevs:application-timestamped>
    <wlevs:expression>inboundTime</wlevs:expression>
  </wlevs:application-timestamped>
</wlevs:channel>
...

```

8. Configure the Oracle Event Processing high availability correlating output adapter failOverDelay.

[Example 24–38](#) shows a correlating output adapter configuration where the failOverDelay is 2000 milliseconds.

#### **Example 24–38 Correlating Output Adapter Configuration: failOverDelay**

```

...
<wlevs:adapter id="myHaCorrelatingAdapter" provider="ha-correlating" >
  <wlevs:listener ref="JMSOutboundAdapter"/>
  <wlevs:instance-property name="failOverDelay" value="2000"/>
</wlevs:adapter>
...

```

For more information, see [Section , "Correlating Output Adapter EPN Assembly File Configuration"](#).

9. Create a second regular JMS input adapter.

[Example 24–39](#) shows a JMS adapter named `JMSInboundAdapter2`.

**Example 24–39 Inbound JMS Adapter Assembly File**

```
...
<wlevs:adapter id="JMSInboundAdapter2" provider="jms-inbound">
</wlevs:adapter>
...
```

This JMS input adapter must be configured identically to the first JMS input adapter (in this example, `JMSInboundAdapter`). [Example 24–40](#) shows the component configuration file for both the JMS input adapters. Note that both have exactly the same configuration, including the same provider.

**Example 24–40 Inbound JMS Adapter Component Configuration File**

```
<?xml version="1.0" encoding="UTF-8"?>
<wlevs:config
  xmlns:wlevs="http://www.bea.com/ns/wlevs/config/application"
  xmlns:ha="http://www.oracle.com/ns/cep/config/cluster">
  ...
  <jms-adapter>
    <name>JMSInboundAdapter</name>
    <jndi-provider-url>t3://localhost:7001</jndi-provider-url>
    <destination-jndi-name>./Topic1</destination-jndi-name>
    <user>weblogic</user>
    <password>weblogic</password>
    <work-manager>JettyWorkManager</work-manager>
    <concurrent-consumers>1</concurrent-consumers>
  </jms-adapter>

  <jms-adapter>
    <name>JMSInboundAdapter2</name>
    <jndi-provider-url>t3://localhost:7001</jndi-provider-url>
    <destination-jndi-name>./Topic1</destination-jndi-name>
    <user>weblogic</user>
    <password>weblogic</password>
    <work-manager>JettyWorkManager</work-manager>
    <concurrent-consumers>1</concurrent-consumers>
  </jms-adapter>
  ...
</wlevs:config>
```

**10. Create a channel to function as the correlated source.**

You must configure this channel with the second regular JMS input adapter as its source.

[Example 24–41](#) shows a correlated source named `clusterCorrelatingOutstream` whose source is `JMSInboundAdapter2`.

**Example 24–41 Creating the Correlated Source**

```
...
<wlevs:adapter id="JMSInboundAdapter2" provider="jms-inbound">
</wlevs:adapter>

<wlevs:channel id="clusterCorrelatingOutstream" event-type="StockTick" advertise="true">
  <wlevs:source ref="JMSInboundAdapter2"/>
</wlevs:channel>
```

**11. Configure the Oracle Event Processing high availability correlating output adapter with the correlated source.**

[Example 24–38](#) shows a correlating output adapter configuration where the `correlatedSource` is `clusterCorrelatingOutstream`.

**Example 24–42 Correlating Output Adapter: `correlatedSource`**

```
...
<wlevs:adapter id="myHaCorrelatingAdapter" provider="ha-correlating" >
  <wlevs:listener ref="JMSOutboundAdapter"/>
  <wlevs:instance-property name="failOverDelay" value="2000"/>
  <wlevs:instance-property name="correlatedSource"
value="clusterCorrelatingOutstream"/>
</wlevs:adapter>
...
```

For more information, see [Section , "Correlating Output Adapter EPN Assembly File Configuration"](#).

12. If your application is an Oracle Event Processing high availability Type 1 application (the application must generate exactly the same sequence of output events as existing secondaries), configure the `warm-up-window-length` for the correlating output adapter.

For more information, see:

- [Section , "Choose an Adequate warm-up-window-length Time"](#)
- [Section , "Correlating Output Adapter Component Configuration File Configuration"](#)

13. Configure the component configuration file to enable `session-transacted` for both inbound JMS adapters and the outbound JMS adapter as [Example 24–43](#) shows:

**Example 24–43 Inbound and Outbound JMS Adapter Component Configuration File**

```
<?xml version="1.0" encoding="UTF-8"?>
<wlevs:config
  xmlns:wlevs="http://www.bea.com/ns/wlevs/config/application"
  xmlns:ha="http://www.oracle.com/ns/cep/config/cluster">
  ...
  <jms-adapter>
    <name>JMSInboundAdapter</name>
    <jndi-provider-url>t3://localhost:7001</jndi-provider-url>
    <destination-jndi-name>./Topic1</destination-jndi-name>
    <user>weblogic</user>
    <password>weblogic</password>
    <work-manager>JettyWorkManager</work-manager>
    <concurrent-consumers>1</concurrent-consumers>
    <session-transacted>true</session-transacted>
  </jms-adapter>

  <jms-adapter>
    <name>JMSInboundAdapter2</name>
    <jndi-provider-url>t3://localhost:7001</jndi-provider-url>
    <destination-jndi-name>./Topic1</destination-jndi-name>
    <user>weblogic</user>
    <password>weblogic</password>
    <work-manager>JettyWorkManager</work-manager>
    <concurrent-consumers>1</concurrent-consumers>
    <session-transacted>true</session-transacted>
  </jms-adapter>
  ...
  <jms-adapter>
    <name>JMSOutboundAdapter</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>
    <session-transacted>true</session-transacted>
  </jms-adapter>
  ...
</wlevs:config>

```

14. Optionally, configure the component configuration file to include the Oracle Event Processing high availability input adapter and correlating output adapter as [Example 24–27](#) shows.

**Example 24–44 High Availability Input and Output Adapter Component Configuration File**

```

<?xml version="1.0" encoding="UTF-8"?>
<wlevs:config
  xmlns:wlevs="http://www.bea.com/ns/wlevs/config/application"
  xmlns:ha="http://www.oracle.com/ns/cep/config/cluster">
  ...
  <ha:ha-inbound-adapter>
    <name>myHaInputAdapter</name>
  </ha:ha-inbound-adapter>
  ...
  <ha:ha-correlating-adapter>
    <name>myHaBroadcastAdapter</name>
    <fail-over-delay>2000</fail-over-delay>
  </ha:ha-correlating-adapter>
  ...
</wlevs:config>

```

For more information, see:

- [Section , "High Availability Input Adapter Component Configuration File Configuration"](#)
  - [Section , "Correlating Output Adapter Component Configuration File Configuration"](#)
15. Optionally, add an `ActiveActiveGroupBean` to your EPN to improve scalability. For more information, see [Section , "Configuring Scalability With the ActiveActiveGroupBean"](#).
16. Deploy your application to the deployment group you created in step 1. For more information, see [Section , "Deploying Oracle Event Processing Applications"](#).

Oracle Event Processing automatically selects one of the Oracle Event Processing servers as the primary.

## Configuring High Availability Adapters

You configure Oracle Event Processing high availability adapters in the EPN assembly file and component configuration files, similar to how you configure other components in the EPN, such as channels or processors. For general information about these configuration files, see:

- [Section , "Overview of EPN Assembly Files"](#)
- [Section , "Overview of Component Configuration Files"](#)

---



---

**Note:** After making any Oracle Event Processing high availability configuration changes, you must redeploy your Oracle Event Processing application. See [Section , "Deploying Oracle Event Processing Applications"](#).

---



---

This section describes the configurable options for each of the Oracle Event Processing high availability adapters, including:

- [Section , "How to Configure the High Availability Input Adapter"](#)
- [Section , "How to Configure the Buffering Output Adapter"](#)
- [Section , "How to Configure the Broadcast Output Adapter"](#)
- [Section , "How to Configure the Correlating Output Adapter"](#)

### How to Configure the High Availability Input Adapter

The Oracle Event Processing high availability broadcast input adapter is implemented by `BroadcastInputAdapter`.

This section describes how to configure the Oracle Event Processing high availability input adapter, including:

- [Section , "High Availability Input Adapter EPN Assembly File Configuration"](#)
- [Section , "High Availability Input Adapter Component Configuration File Configuration"](#)

For more information, see [Section , "High Availability Input Adapter"](#).

**High Availability Input Adapter EPN Assembly File Configuration** The root element for declaring an Oracle Event Processing high availability input adapter is `wlevs:adapter` with `provider` element set to `ha-inbound` as [Example 24–45](#) shows. You specify a `wlevs:listener` element for the Oracle Event Processing high availability input adapter in the actual input adapter as [Example 24–45](#) shows.

#### **Example 24–45 High Availability Input Adapter EPN Assembly File**

```
<wlevs:adapter id="jmsAdapter" provider="jms-inbound"
  <wlevs:listener ref="myHaInputAdapter" />
</wlevs:adapter>

<wlevs:adapter id="myHaInputAdapter" provider="ha-inbound">
  <wlevs:instance-property name="keyProperties" value="id" />
  <wlevs:instance-property name="timeProperty" value="arrivalTime" />
  <wlevs:instance-property name="eventType" value="MyEventType" />
</wlevs:adapter>

<wlevs:channel id="inputChannel" event-type="MyEventType ">
  <wlevs:source ref="myHaInputAdapter" />
  <wlevs:application-timestamped>
    <wlevs:expression>arrivalTime</wlevs:expression>
  </wlevs:application-timestamped>
</wlevs:channel>
```

[Table 24–3](#) describes the additional child elements of `wlevs:adapter` you can configure for an Oracle Event Processing high availability input adapter.

**Table 24–3 Child Elements of wlevs:adapter for the High Availability Input Adapter**

Child Element	Description
wlevs:instance-property	Specify one or more instance-property element name and value attributes as <a href="#">Table 24–4</a> describes.

[Table 24–4](#) lists the instance properties that the Oracle Event Processing high availability input adapter supports.

**Table 24–4 High Availability Input Adapter Instance Properties**

Name	Value
timeProperty	Specify the name of the event property to which the high availability input adapter assigns a time value. This is the same property that you use in the wlevs:application-timestamped element of the downstream EPN component to which the high availability input adapter is connected as <a href="#">Example 24–45</a> shows.
keyProperties	Specify a space delimited list of one or more event properties that the Oracle Event Processing high availability input adapter uses to identify event instances. If you specify more than one event property, you must specify a <a href="#">keyClass</a> . Default: all event properties.
keyClass	Specify the fully qualified class name of a Java class used as a compound key. By default, all JavaBean properties in the keyClass are assumed to be keyProperties, unless the keyProperties setting is used.
eventType	Specify the type name of the events that the Oracle Event Processing high availability input adapter receives from the actual input adapter. This is the same event type that you use in the downstream EPN component to which the high availability input adapter is connected as <a href="#">Example 24–45</a> shows. For tuple events, this property is mandatory. For all other Java class-based event types, this property is optional. For more information, see <a href="#">Section , "Overview of Oracle Event Processing Event Types"</a> .

**High Availability Input Adapter Component Configuration File Configuration** The root element for configuring an Oracle Event Processing high availability input adapter is ha-inbound-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 as [Example 24–50](#) shows.

**Example 24–46 High Availability Input Adapter Component Configuration File**

```
<ha:ha-inbound-adapter>
  <name>myHaInputAdapter</name>
  <heartbeat units="millis">1000</heartbeat>
  <batch-size>10</batch-size>
</ha:ha-inbound-adapter>
```

[Table 24–5](#) describes the additional child elements of ha-inbound-adapter you can configure for an Oracle Event Processing high availability input adapter.

**Table 24–5 Child Elements of ha-inbound-adapter for the High Availability Input Adapter**

Child Element	Description
heartbeat	Specify the length of time that the Oracle Event Processing high availability input adapter can be idle before it generates a heartbeat event to advance time as an integer number of units. Valid values for attribute units: <ul style="list-style-type: none"> <li>■ nanos: wait the specified number of nanoseconds.</li> <li>■ millis: wait the specified number of milliseconds.</li> <li>■ secs: wait the specified number of seconds.</li> </ul> Default: Heartbeats are not sent.
batch-size	Specify the number of events in each timing message that the primary broadcasts to its secondaries. A value of n means that n {key, time} pairs are sent in each message. You can use this property for performance tuning (see <a href="#">Section , "High Availability Input Adapter Configuration"</a> ) Default: 1 (disable batching).

### How to Configure the Buffering Output Adapter

The Oracle Event Processing high availability buffering output adapter is implemented by `SlidingWindowQueueTrimmingAdapter`.

This section describes how to configure the Oracle Event Processing high availability buffering output adapter, including:

- [Section , "Buffering Output Adapter EPN Assembly File Configuration"](#)
- [Section , "Buffering Output Adapter Component Configuration File Configuration"](#)

For more information, see [Section , "Buffering Output Adapter"](#).

**Buffering Output Adapter EPN Assembly File Configuration** The root element for declaring an Oracle Event Processing high availability buffering output adapter is `wlevs:adapter` with `provider` element set to `ha-buffering` as [Example 24–47](#) shows.

**Example 24–47 Buffering Output Adapter EPN Assembly File**

```
<wlevs:adapter id="mySlidingWindowingAdapter" provider="ha-buffering">
  <wlevs:listener>
    <bean class="com.bea.wlevs.example.cluster.ClusterAdapterBean"/>
  </wlevs:listener>
  <wlevs:instance-property name="windowLength" value="15000"/>
</wlevs:adapter>
```

[Table 24–6](#) describes the additional child elements of `wlevs:adapter` you can configure for an Oracle Event Processing high availability buffering output adapter.

**Table 24–6 Child Elements of wlevs:adapter for the Buffering Output Adapter**

Child Element	Description
wlevs:listener	Specify the regular output adapter downstream from this Oracle Event Processing high availability buffering output adapter.
wlevs:instance-property	Specify one or more <code>instance-property</code> element name and value attributes as <a href="#">Table 24–7</a> describes.

[Table 24–7](#) lists the instance properties that the Oracle Event Processing high availability broadcast output adapter supports.



**Table 24–7 Buffering Output Adapter Instance Properties**

Name	Value
windowLength	Specify the size of the sliding window as an integer number of milliseconds. Default: 15000.

**Buffering Output Adapter Component Configuration File Configuration** The root element for configuring an Oracle Event Processing high availability buffering output adapter is `ha-buffering-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 as [Example 24–48](#) shows.

**Example 24–48 Buffering Output Adapter Component Configuration File**

```
<ha:ha-buffering-adapter >
  <name>mySlidingWindowingAdapter</name>
  <window-length>15000</window-length>
  <warm-up-window-length units="minutes">6</warm-up-window-length>
</ha:ha-buffering-adapter >
```

[Table 24–8](#) describes the additional child elements of `ha-buffering-adapter` you can configure for an Oracle Event Processing high availability buffering output adapter.

**Table 24–8 Child Elements of ha-buffering-adapter for the Buffering Output Adapter**

Child Element	Description
window-length	Specify the size of the sliding window as an integer number of milliseconds. Default: 15000.
warm-up-window-length	Specify the length of time it takes the application to rebuild state after a previously failed secondary restarts or a new secondary is added as an integer number of units. Valid values for attribute units: <ul style="list-style-type: none"> <li>▪ seconds: wait the specified number of seconds.</li> <li>▪ minutes: wait the specified number of minutes.</li> </ul> Default: units is seconds. For more information, see <a href="#">Section , "Choose an Adequate warm-up-window-length Time"</a> .

## How to Configure the Broadcast Output Adapter

The Oracle Event Processing high availability broadcast output adapter is implemented by class `GroupBroadcastQueueTrimmingAdapter`.

This section describes how to configure the Oracle Event Processing high availability broadcast output adapter, including:

- [Section , "Broadcast Output Adapter EPN Assembly File Configuration"](#)
- [Section , "Broadcast Output Adapter Component Configuration File Configuration"](#)

For more information, see [Section , "Broadcast Output Adapter"](#).

**Broadcast Output Adapter EPN Assembly File Configuration** The root element for declaring an Oracle Event Processing high availability broadcast output adapter is `wlevs:adapter` with `provider` element set to `ha-broadcast` as [Example 24–49](#) shows.

**Example 24–49 Broadcast Output Adapter EPN Assembly File**

```
<wlevs:adapter id="myBroadcastAdapter" provider="ha-broadcast">
  <wlevs:listener ref="actualAdapter"/>
  <wlevs:instance-property name="keyProperties" value="time"/>
  <wlevs:instance-property name="monotonic" value="true"/>
</wlevs:adapter>
```

Table 24–9 describes the additional child elements of `wlevs:adapter` you can configure for an Oracle Event Processing high availability broadcast output adapter.

**Table 24–9 Child Elements of `wlevs:adapter` for the Broadcast Output Adapter**

Child Element	Description
<code>wlevs:listener</code>	Specify the regular output adapter downstream from this Oracle Event Processing high availability broadcast output adapter.
<code>wlevs:instance-property</code>	Specify one or more <code>instance-property</code> element name and value attributes as Table 24–10 describes.

Table 24–10 lists the instance properties that the Oracle Event Processing high availability broadcast output adapter supports.

**Table 24–10 Broadcast Output Adapter Instance Properties**

Name	Value
<code>keyProperties</code>	Specify a space delimited list of one or more event properties that the Oracle Event Processing high availability broadcast output adapter uses to identify event instances.  If you specify more than one event property, you must specify a <code>keyClass</code> .  Default: all event properties.
<code>keyClass</code>	Specify the fully qualified class name of a Java class used as a compound key.  By default, all JavaBean properties in the <code>keyClass</code> are assumed to be <code>keyProperties</code> , unless the <code>keyProperties</code> setting is used.  A compound key may be <code>monotonic</code> and may be <code>totalOrder</code> .
<code>monotonic</code>	Specify whether or not the key value is constantly increasing (like a time value).  Valid values are: <ul style="list-style-type: none"> <li>▪ <code>true</code>: the key is constantly increasing.</li> <li>▪ <code>false</code>: the key is not constantly increasing.</li> </ul> Default: <code>false</code> .
<code>totalOrder</code>	Specify whether or not event keys are unique. Applicable only when instance property <code>monotonic</code> is set to <code>true</code> .  Valid values are: <ul style="list-style-type: none"> <li>▪ <code>true</code>: event keys are unique.</li> <li>▪ <code>false</code>: event keys are not unique.</li> </ul> Default: <code>true</code> .

**Broadcast Output Adapter Component Configuration File Configuration** The root element for configuring an Oracle Event Processing high availability broadcast output adapter is `ha-broadcast-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 as Example 24–50 shows.

**Example 24–50 Broadcast Output Adapter Component Configuration File**

```
<ha:ha-broadcast-adapter>
  <name>myBroadcastAdapter</name>
  <trimming-interval units="events">10</trimming-interval>
  <warm-up-window-length units="minutes">6</warm-up-window-length>
</ha:ha-broadcast-adapter>
```

[Table 24–11](#) describes the additional child elements of `ha-broadcast-adapter` you can configure for an Oracle Event Processing high availability broadcast output adapter.

**Table 24–11 Child Elements of `ha-broadcast-adapter` for the Broadcast Output Adapter**

Child Element	Description
trimming-interval	<p>Specify the interval at which trimming messages are broadcast as an integer number of units. You can use this property for performance tuning (see <a href="#">Section , "Broadcast Output Adapter Configuration"</a>).</p> <p>Valid values for attribute units:</p> <ul style="list-style-type: none"> <li>▪ <code>events</code>: broadcast trimming messages after the specified number of milliseconds.</li> <li>▪ <code>millis</code>: broadcast trimming messages after the specified number of events are processed.</li> </ul> <p>Default: units is events.</p>
warm-up-window-length	<p>Specify the length of time it takes the application to rebuild state after a previously failed secondary restarts or a new secondary is added as an integer number of units.</p> <p>Valid values for attribute units:</p> <ul style="list-style-type: none"> <li>▪ <code>seconds</code>: wait the specified number of seconds.</li> <li>▪ <code>minutes</code>: wait the specified number of minutes.</li> </ul> <p>Default: units is seconds.</p> <p>For more information, see <a href="#">Section , "Choose an Adequate warm-up-window-length Time"</a>.</p>

**How to Configure the Correlating Output Adapter**

The Oracle Event Processing high availability correlating output adapter is implemented by class `CorrelatedQueueTrimmingAdapter`.

This section describes how to configure the Oracle Event Processing high availability correlating output adapter, including:

- [Section , "Correlating Output Adapter EPN Assembly File Configuration"](#)
- [Section , "Correlating Output Adapter Component Configuration File Configuration"](#)

For more information, see [Section , "Correlating Output Adapter"](#).

**Correlating Output Adapter EPN Assembly File Configuration** The root element for declaring an Oracle Event Processing high availability correlating output adapter is `wlevs:adapter` with provider element set to `ha-correlating` as [Example 24–51](#) shows.

**Example 24–51 Correlating Output Adapter EPN Assembly File**

```
<wlevs:adapter id="myCorrelatingAdapter" provider="ha-correlating">
  <wlevs:listener>
    <bean class="com.bea.wlevs.example.cluster.ClusterAdapterBean"/>
  </wlevs:listener>
  <wlevs:instance-property name="correlatedSource" ref="clusterCorrOutstream"/>
```

```
<wlevs:instance-property name="failOverDelay" value="2000"/>
</wlevs:adapter>
```

Table 24–12 describes the additional child elements of `wlevs:adapter` you can configure for an Oracle Event Processing high availability correlating output adapter.

**Table 24–12 Child Elements of `wlevs:adapter` for the Correlating Output Adapter**

Child Element	Description
<code>wlevs:listener</code>	Specify the regular output adapter downstream from this Oracle Event Processing high availability buffering output adapter.
<code>wlevs:instance-property</code>	Specify one or more <code>instance-property</code> element name and value attributes as Table 24–13 describes.

Table 24–13 lists the instance properties that the Oracle Event Processing high availability correlating output adapter supports.

**Table 24–13 Correlating Output Adapter Instance Properties**

Name	Value
<code>correlatedSource</code>	Specify the event source that will be used to correlate against. Events seen from this source will be purged from the trimming queue. Events still in the queue at failover will be replayed.
<code>failOverDelay</code>	Specify the delay timeout in milliseconds that is used to decide how soon after failover correlation should restart. Default: 0 ms.

**Correlating Output Adapter Component Configuration File Configuration** The root element for configuring an Oracle Event Processing high availability correlating output adapter is `ha-correlating-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 as Example 24–52 shows.

**Example 24–52 Correlating Output Adapter Component Configuration File**

```
<ha:ha-correlating-adapter>
  <name>myCorrelatingAdapter</name>
  <window-length>15000</window-length>
  <warm-up-window-length units="minutes">6</warm-up-window-length>
</ha:ha-correlating-adapter>
```

Table 24–14 describes the additional child elements of `ha-broadcast-adapter` you can configure for an Oracle Event Processing high availability correlating output adapter.

**Table 24–14 Child Elements of `ha-correlating-adapter` for the Correlating Output Adapter**

Child Element	Description
<code>fail-over-delay</code>	Specify the delay timeout in milliseconds that is used to decide how soon after failover correlation should restart. Default: 0 ms.

**Table 24–14 (Cont.) Child Elements of ha-correlating-adapter for the Correlating Output Adapter**

Child Element	Description
warm-up-window-length	<p>Specify the length of time it takes the application to rebuild state after a previously failed secondary restarts or a new secondary is added as an integer number of units.</p> <p>Valid values for attribute units:</p> <ul style="list-style-type: none"><li>■ seconds: wait the specified number of seconds.</li><li>■ minutes: wait the specified number of minutes.</li></ul> <p>Default: units is seconds.</p> <p>For more information, see <a href="#">Section , "Choose an Adequate warm-up-window-length Time"</a>.</p>



---

---

## Developing Scalable Applications

This chapter introduces components and design patterns that you can use to allow your Oracle Event Processing applications to scale with an increasing event load, along with how to configure scalability for your application, including information on setting up event partitioning.

This chapter includes the following sections:

- [Understanding Scalability](#)
- [Configuring Scalability](#)

### Understanding Scalability

This section introduces components and design patterns that you can use to allow your Oracle Event Processing applications to scale with an increasing event load.

This section includes the following sections:

- [Section , "Scalability Options"](#)
- [Section , "Scalability Components"](#)

### Scalability Options

Oracle Event Processing provides options that you can use to allow your Oracle Event Processing application to scale with an increasing event load.

In general, you can design your application to partition an input event stream and process events in parallel at the point of event ingress, within the Event Processing Network (EPN), or both.

You should plan to use scalability and parallel processing as early in the event processing sequence as possible. For example, parallel process high-volume, low-value events to extract the typically low-volume, high-value events your application depends on.

Oracle Event Processing provides a variety of scalability components you can use as [Section , "Scalability Components"](#) describes.

### Scalability and High Availability

Because scalability often involves deploying an application to multiple servers, it is advantageous to also consider high availability options when designing your Oracle Event Processing application for scalability.

Input stream partitioning and parallel processing impose important restrictions on application design, such as preserving event order and carefully managing the use of multi-threading.

For more information on high availability options, see:

- [Section , "High Availability Architecture"](#)
- [Section , "Designing an Oracle Event Processing Application for High Availability"](#)

## Scalability Components

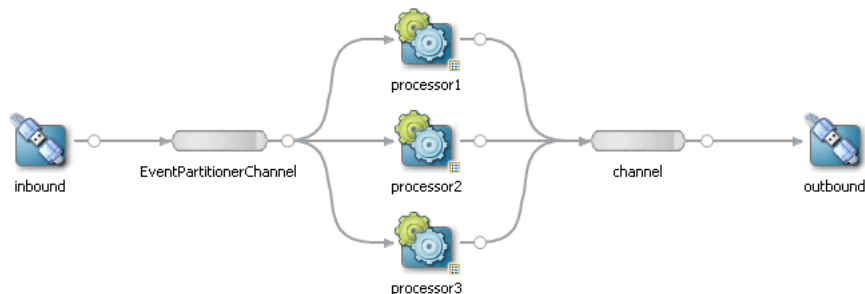
Oracle Event Processing provides the following components that you can use to improve the scalability of your Oracle Event Processing applications:

- [Section , "EventPartitioner"](#)
- [Section , "ActiveActiveGroupBean"](#)

### EventPartitioner

A `com.bea.wlevs.channel.EventPartitioner` provides a mechanism for partitioning events on a channel across its output event sinks as [Figure 25–1](#) shows.

**Figure 25–1 Event Partitioner EPN**



This section describes:

- [Section , "EventPartitioner Implementation"](#)
- [Section , "EventPartitioner Initialization"](#)
- [Section , "EventPartitioner Threading"](#)
- [Section , "EventPartitioner Restrictions"](#)

For more information, see [Section , "Configuring Scalability With a Channel EventPartitioner"](#).

**EventPartitioner Implementation** Oracle Event Processing provides a default event property-based `EventPartitioner` that you can configure a channel to use.

When you configure a channel to use the default `EventPartitioner`, you specify the name of an event property by which the channel partitions events. The default `EventPartitioner` calculates a hash key using the event property value's `Object.hashCode()` as input to an internal hash function. The `hashkey % number-of-listeners` is used to calculate which listener will receive the event. This algorithm is based on the same algorithm used by `HashMap` to calculate in which bucket to place a new item. In practice, this means events with the same event property value are sent to the same listener.



---



---

**Note:** The default event property-based `EventPartitioner` does not dispatch in Round Robin fashion.

---



---

Optionally, you can create your own event partitioner instance and configure a channel to use it instead to customize how events are dispatched to the channel's listeners.

For more information, see:

- [Section , "How to Configure Scalability With the Default Channel EventPartitioner"](#)
- [Section , "How to Configure Scalability With a Custom Channel EventPartitioner"](#)

**EventPartitioner Initialization** By default, the Oracle Event Processing server initializes each event partitioner on deployment and will re-initialize event partitioners on re-deployment by invoking the `EventPartitioner` method `activateConfiguration` is before `ActivatableBean.afterConfigurationActive` and before your `EventPartitioner` class's `partition` method is invoked.

**EventPartitioner Threading** [Table 25–1](#) lists the threading options you can use with an event partitioner channel.

**Table 25–1 Event Partitioner Channel Threading Options**

Threads Allocated In	Description	When to Use
Channel	<ul style="list-style-type: none"> <li>■ Channel <code>max-threads</code> set to the number of listeners.</li> </ul>	<p>Usually acceptable if conversion of the external message format into the internal event format is inexpensive.</p> <p>Lets the multithreading be controlled at the channel granularity. Some channels may require a higher number of threads than others due to differences in volume.</p>
Adapter	<ul style="list-style-type: none"> <li>■ Channel <code>max-threads</code> set to 0.</li> <li>■ Implement a multi-threaded adapter with at least one thread per partition listener.</li> </ul>	<p>Usually preferable if conversion of the external message format into the internal event format is expensive.</p> <p>This approach is best when the adapter is multithreaded and the inbound event rate is high enough that the adapter becomes a bottleneck unless multiple threads can be used to scale the inbound processing.</p>

---



---

**Note:** In either approach, event order cannot be guaranteed. This is true whenever multiple threads are used.

---



---

**EventPartitioner Restrictions** When configuring a channel to use an event partitioner, consider the following restrictions:

- Batching is not supported when you configure a channel with an event partitioner.
- For more information, [Section , "Batch Processing Channels"](#).

### ActiveActiveGroupBean

Using the `com.oracle.cep.cluster.hagroups.ActiveActiveGroupBean`, you can partition an incoming JMS stream in Oracle Event Processing applications by utilizing the notification groups that the `ActiveActiveGroupBean` creates.

You add an `ActiveActiveGroupBean` to your EPN assembly file as [Example 25–1](#) shows.

**Example 25–1 ActiveActiveGroupBean bean Element**

```
<bean id="clusterAdapter" class="com.oracle.cep.cluster.hagroups.ActiveActiveGroupBean">
</bean>
```

By default, the `ActiveActiveGroupBean` creates notification groups named:

```
ActiveActiveGroupBean_X
```

Where *X* is a string.

At runtime, the `ActiveActiveGroupBean` scans the existing groups defined on the Oracle Event Processing server and applies a default pattern match of:

```
ActiveActiveGroupBean_\\w+
```

When it finds a match, it creates a notification group of that name.

Optionally, you can define your own cluster group pattern match as [Section , "How to Configure the ActiveActiveGroupBean Group Pattern Match"](#) describes.

This section describes:

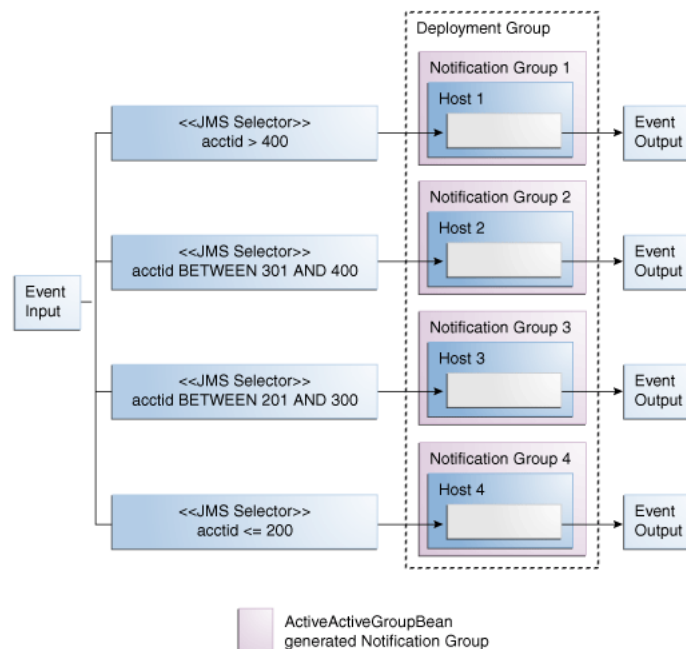
- [Section , "Scalability in an Oracle Event Processing Application Using the ActiveActiveGroupBean Without High Availability"](#)
- [Section , "Scalability in an Oracle Event Processing Application Using the ActiveActiveGroupBean With High Availability"](#)

For more information, see:

- [Section , "Deployment Group and Notification Group"](#)
- [Section , "Configuring Scalability With the ActiveActiveGroupBean"](#)

**Scalability in an Oracle Event Processing Application Using the ActiveActiveGroupBean Without High Availability** You can use the `ActiveActiveGroupBean` to partition an incoming JMS event stream by selector in an Oracle Event Processing application that is not configured for high availability.

Consider the multi-server domain that [Figure 25–2](#) shows.

**Figure 25–2 Oracle Event Processing ActiveActiveGroupBean Without High Availability**

In this scalability scenario, you define a cluster group in the Oracle Event Processing server `config.xml` on each server (`ActiveActiveGroupBean_group1` on Host 1, `ActiveActiveGroupBean_group2` on Host 2, and so on) and add an instance of the `ActiveActiveGroupBean` to your Oracle Event Processing application to define notification groups based on these cluster groups.

Each notification group is bound to a different JMS selector. The component configuration file in your Oracle Event Processing application contains the same `jms-adapter` configuration as [Example 25–2](#) shows.

#### **Example 25–2 Common `jms-adapter` Selector Definitions**

```
<jms-adapter>
  <message-selector>${CONDITION}</message-selector>
  <bindings>
    <group-binding group-id="ActiveActiveGroupBean_group1">
      <param id="CONDITION">acctid > 400</param>
    </group-binding>
    <group-binding group-id="ActiveActiveGroupBean_group2">
      <param id="CONDITION">acctid BETWEEN 301 AND 400</param>
    </group-binding>
    <group-binding group-id="ActiveActiveGroupBean_group3">
      <param id="CONDITION">acctid BETWEEN 201 AND 300</param>
    </group-binding>
    <group-binding group-id="ActiveActiveGroupBean_group4">
      <param id="CONDITION">acctid <= 200</param>
    </group-binding>
  </bindings>
</jms-adapter>
```

At runtime, the `ActiveActiveGroupBean` instance in each Oracle Event Processing application instance on each Oracle Event Processing server finds its `ActiveActiveGroupBean_` cluster group and creates a notification group based on it. The Oracle Event Processing application then configures itself with the

message-selector that corresponds to the group-id that matches that notification group. This partitions the JMS topic so that each instance of App1 processes a subset of the total number of messages in parallel.

---

**Note:** Within each instance of App1, you could further increase parallel processing by configuring an event partitioner channel as [Section , "EventPartitioner"](#) describes.

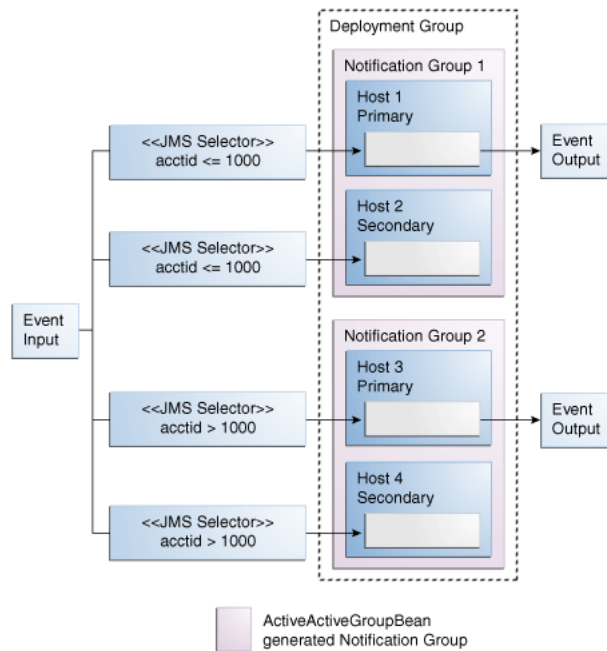
---

For more information, see [Section , "How to Configure Scalability in a JMS Application Without Oracle Event Processing High Availability"](#).

**Scalability in an Oracle Event Processing Application Using the ActiveActiveGroupBean With High Availability** In addition to partitioning an incoming JMS event stream by selector, you can also use the ActiveActiveGroupBean to configure two or more Oracle Event Processing servers to function as a single, high availability unit.

Consider the multi-server domain with an Oracle Event Processing high availability application deployed to it that [Figure 25-3](#) shows.

**Figure 25-3 Oracle Event Processing ActiveActiveGroupBean With High Availability**



In this scenario, you create the same ActiveActiveGroupBean\_ cluster group on Host 1 and Host 2 (ActiveActiveGroupBean\_group1) and the same ActiveActiveGroupBean\_ cluster group on Host 3 and Host 4 (ActiveActiveGroupBean\_group2).

At runtime, the ActiveActiveGroupBean instance in each Oracle Event Processing application instance on each Oracle Event Processing server finds its ActiveActiveGroupBean\_ cluster group and creates a notification group based on it. Both Host 1 and Host 2 belong to one notification group (ActiveActiveGroupBean\_group1) and both Host 3 and Host 4 belong to another notification group (ActiveActiveGroupBean\_group2).

Each Oracle Event Processing application then configures itself with the message-selector that corresponds to the group-id that matches that notification

group. This partitions the JMS topic so that each instance of App1 processes a subset of the total number of messages in parallel.

When more than one Oracle Event Processing server belongs to the same notification group, the `ActiveActiveGroupBean` ensures that only the primary server in each notification group outputs events. Within a given notification group, should the primary server go down, then an Oracle Event Processing high availability fail over occurs and one of the secondary servers in that notification group is declared the new primary and resumes outputting events according to the Oracle Event Processing high availability quality of service you configure.

---

**Note:** Within each instance of App1, you could further increase parallel processing by configuring an event partitioner channel as [Section , "EventPartitioner"](#) describes.

---

For more information, see [Section , "How to Configure Scalability in a JMS Application With Oracle Event Processing High Availability"](#).

## Configuring Scalability

This section describes how to configure scalability for your Oracle Event Processing application, including information on setting up event partitioning.

This section includes the following sections:

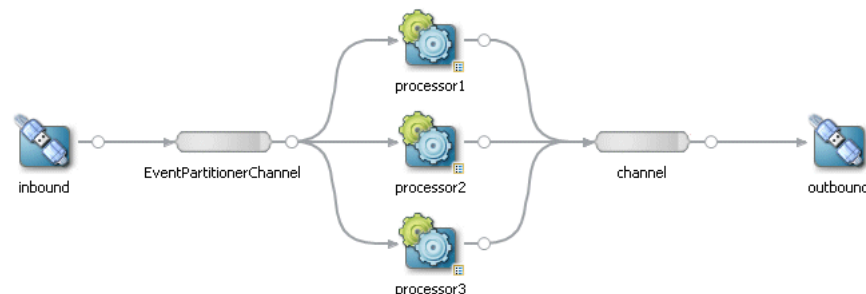
- [Section , "Configuring Scalability With a Channel EventPartitioner"](#)
- [Section , "Configuring Scalability With the ActiveActiveGroupBean"](#)

### Configuring Scalability With a Channel EventPartitioner

This section describes how to configure a channel with an Oracle Event Processing event partitioner as [Figure 25-4](#) shows, including:

- [Section , "How to Configure Scalability With the Default Channel EventPartitioner"](#)
- [Section , "How to Configure Scalability With a Custom Channel EventPartitioner"](#)

**Figure 25-4** *EventPartitioner EPN*



In this example, assume that the inbound adapter is sending events of type `PriceEvent`, defined as [Example 25-3](#) shows:

**Example 25-3** *Definition of Event Type PriceEvent*

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

```

<wlevs:event-type type-name="PriceEvent">
  <wlevs:properties>
    <wlevs:property name="symbol" type="char" />
    <wlevs:property name="price" type="long" />
  </wlevs:properties>
</wlevs:event-type>
</wlevs:event-type-repository>

```

For more information, see [Section , "EventPartitioner"](#).

### How to Configure Scalability With the Default Channel EventPartitioner

You can configure a channel to use the default event property-based event partitioner. Each time an incoming event arrives, the channel selects a listener and dispatches the event to that listener instead of broadcasting each event to every listener.

Optionally, you can implement your own `EventPartitioner` class to customize how the channel dispatches events to its listeners as [Section , "How to Configure Scalability With a Custom Channel EventPartitioner"](#) describes.

#### To configure scalability with the default channel EventPartitioner:

1. Add a channel to your EPN.

In [Figure 25-4](#), the channel is `EventPartitionerChannel`.

For more information, see [Chapter 10, "Connecting EPN Stages Using Channels"](#).

2. Connect the channel to an upstream adapter.

In [Figure 25-4](#), the upstream adapter is `inbound`.

For more information, see [Section , "Connecting Nodes"](#).

3. Connect the channel to two or more listeners.

In [Figure 25-4](#), the channel is connected to Oracle CQL processors `processor1`, `processor2`, and `processor3`.

For more information, see [Section , "Connecting Nodes"](#)

4. Edit the EPN assembly file to add a `partitionByEventProperty` instance property to the channel element.

The value of this instance-property is the name of the event property by which the channel partitions events.

In this example, the channel partitions events by event property `symbol`.

```

...
<wlevs:channel id="EventPartitionerChannel" event-type="PriceEvent">
  <wlevs:instance-property name="partitionByEventProperty" value="symbol" />
/>
  <wlevs:listener ref="processor1" />
  <wlevs:listener ref="processor2" />
  <wlevs:listener ref="processor3" />
  <wlevs:source ref="inbound" />
</wlevs:channel>
...

```

For more information, see [Section , "EventPartitioner Implementation"](#).

5. Decide how you want Oracle Event Processing to allocate threads as [Section , "EventPartitioner Threading"](#) describes:
  - a. If you want the channel to allocate threads:

- Edit the EPN assembly file to configure the channel to set `max-threads` to the number of listeners.

```
...
    <wlevs:channel id="EventPartitionerChannel" event-type="PriceEvent"
      max-threads="3" >
      <wlevs:instance-property name="eventPartitioner" value="true"
    />
    <wlevs:listener ref="processor1" />
    <wlevs:listener ref="processor2" />
    <wlevs:listener ref="processor3" />
    <wlevs:source ref="inbound" />
  </wlevs:channel>
...
```

**b.** If you want the upstream adapter to allocate threads:

- Edit the EPN assembly file to configure the channel to set `max-threads` to 0.

```
...
    <wlevs:channel id="EventPartitionerChannel" event-type="PriceEvent"
      max-threads="0" >
      <wlevs:instance-property name="eventPartitioner" value="true"
    />
    <wlevs:listener ref="processor1" />
    <wlevs:listener ref="processor2" />
    <wlevs:listener ref="processor3" />
    <wlevs:source ref="inbound" />
  </wlevs:channel>
...
```

- Edit the Oracle Event Processing server `config.xml` file to add a `work-manager` element.

If this work manager is shared by more than one component (such as other adapters and Jetty), then set the `min-threads-constraint` and `max-threads-constraint` elements each to a value greater than the number of listeners.

If this work manager is not shared by more than one component (that is, it is dedicated to the upstream adapter in this configuration), then set the `min-threads-constraint` and `max-threads-constraint` elements equal to the number of listeners.

```
...
<work-manager>
  <name>adapterWorkManager</name>
  <min-threads-constraint>3</min-threads-constraint>
  <max-threads-constraint>3</max-threads-constraint>
</work-manager>
...
```

For more information, see [Section , "work-manager"](#).

- Edit the component configuration file to configure the upstream adapter with this work-manager.

```
...
<adapter>
  <name>inbound</name>
  <work-manager-name>adapterWorkManager</work-manager-name>
</adapter>
```

```
...  
</adapter>  
...
```

6. Assemble and deploy your application.

For more information, see [Chapter 23, "Assembling and Deploying Oracle Event Processing Applications"](#).

At runtime, the channel uses the default event property-based `EventPartitioner` to determine how to dispatch each incoming event to its listeners.

### How to Configure Scalability With a Custom Channel `EventPartitioner`

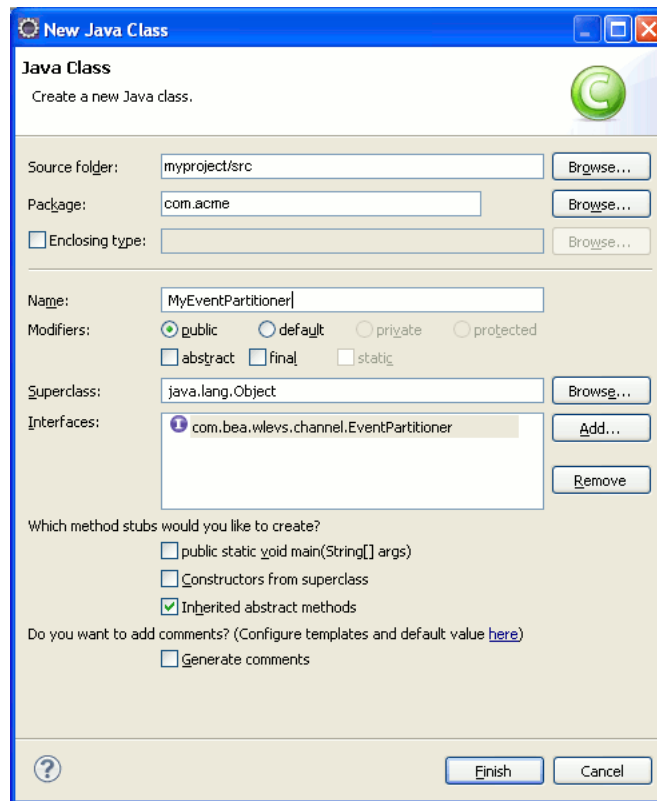
You can implement your own `EventPartitioner` class to customize how a channel dispatches events to its listeners.

Optionally, you can use the default event property-based `EventPartitioner` as [Section , "How to Configure Scalability With the Default Channel `EventPartitioner`"](#) describes.

#### To configure scalability with a custom channel `EventPartitioner`:

1. Using the Oracle Event Processing IDE for Eclipse, open your Oracle Event Processing project.  
For more information, see [Section , "Creating Oracle Event Processing Projects"](#).
2. Edit your `MANIFEST.MF` to import package `com.bea.wlevs.channel`.  
For more information, see [Section , "How to Import a Package"](#).
3. Right-click your project's `src` folder and select **New > Class**.  
The New Java Class dialog appears as [Figure 25–5](#) shows.



**Figure 25–5** New Java Class Dialog

4. Configure the New Java Class dialog as [Table 25–2](#) describes.

**Table 25–2** New Java Class Options for EventPartitioner

Option	Description
Package	Enter the class's package name.
Name	Enter the class's name.
Interfaces	Click Add and use the Implemented Interfaces Selection dialog to locate the <code>com.bea.wlevs.channel.EventPartitioner</code> interface, select it in the <b>Matching Items</b> list, and then click <b>OK</b> .

5. Click **Finish**.

A new `EventPartitioner` class is created as [Example 25–4](#) shows.

**Example 25–4** EventPartitioner Class

```
package com.acme;

import com.bea.wlevs.channel.EventPartitioner;
import com.bea.wlevs.ede.api.EventProcessingException;
import com.bea.wlevs.ede.api.EventType;

public class MyEventPartitioner implements EventPartitioner {

    @Override
    public void activateConfiguration(int arg0, EventType arg1) {
        // TODO Auto-generated method stub
    }
}
```

```

@Override
public int partition(Object arg0) throws EventProcessingException {
    // TODO Auto-generated method stub
    return 0;
}
}

```

6. Complete the implementation of your EventPartitioner as [Example 25–5](#) shows.

#### **Example 25–5 EventPartitioner Class Implementation**

```

package com.acme;

import com.bea.wlevs.channel.EventPartitioner;
import com.bea.wlevs.ede.api.EventProcessingException;
import com.bea.wlevs.ede.api.EventType;

public class MyEventPartitioner implements EventPartitioner {

    private final EventType eventType;
    private int numberOfPartitions;

    @Override
    public void activateConfiguration(int numberOfPartitions, EventType eventType) {
        this.numberOfPartitions = numberOfPartitions;
        this.eventType = eventType;
    }

    @Override
    public int partition(Object event) throws EventProcessingException {
        int dispatchToListener = 0;
        ... // Your implementation.
        return dispatchToListener;
    }
}

```

The `activateConfiguration` method is a callback that the Oracle Event Processing server invokes before `ActivatableBean.afterConfigurationActive` and before your `EventPartitioner` class's `partition` method is invoked.

When you associate this `EventPartitioner` with a channel, the channel will invoke your `EventPartitioner` class's `partition` method each time the channel receives an event.

Your `partition` method must return the index of the listener to which the channel should dispatch the event. The index must be an `int` between 0 and `numberOfPartitions - 1`.

7. Add a channel to your EPN.

In [Figure 25–4](#), the channel is `EventPartitionerChannel`.

For more information, see [Chapter 10, "Connecting EPN Stages Using Channels"](#).

8. Connect the channel to an upstream adapter.

In [Figure 25–4](#), the upstream adapter is `inbound`.

For more information, see [Section , "Connecting Nodes"](#).

9. Connect the channel to two or more listeners.

In [Figure 25-4](#), the channel is connected to Oracle CQL processors `processor1`, `processor2`, and `processor3`.

If you want the channel to perform load balancing, each listener must be identical.

For more information, see:

- [Section , "Connecting Nodes"](#)

10. Edit the EPN assembly file to add an `eventPartitioner` instance property to the channel element.

The value of this `instance-property` is the fully qualified class name of the `EventPartitioner` instance the channel will use to partition events.

This class must be on your Oracle Event Processing application class path.

In this example, the channel uses `EventPartitioner` instance `com.acme.MyEventPartitioner` to partition events.

```
...
    <wlevs:channel id="EventPartitionerChannel" event-type="PriceEvent"
max-threads="0" >
        <wlevs:instance-property name="eventPartitioner"
            value="com.acme.MyEventPartitioner" />
        <wlevs:listener ref="filterFanoutProcessor1" />
        <wlevs:listener ref="filterFanoutProcessor2" />
        <wlevs:listener ref="filterFanoutProcessor3" />
        <wlevs:source ref="PriceAdapter" />
    </wlevs:channel>
...
```

11. Decide how you want Oracle Event Processing to allocate threads as [Section , "EventPartitioner Threading"](#) describes:

- a. If you want the channel to allocate threads:

- Edit the EPN assembly file to configure the channel to set `max-threads` to the number of listeners.

```
...
    <wlevs:channel id="EventPartitionerChannel" event-type="PriceEvent"
        max-threads="3" >
        <wlevs:instance-property name="eventPartitioner" value="true"
/>
        <wlevs:listener ref="processor1" />
        <wlevs:listener ref="processor2" />
        <wlevs:listener ref="processor3" />
        <wlevs:source ref="inbound" />
    </wlevs:channel>
...
```

- b. If you want the upstream adapter to allocate threads:

- Edit the EPN assembly file to configure the channel to set `max-threads` to 0.

```
...
    <wlevs:channel id="EventPartitionerChannel" event-type="PriceEvent"
        max-threads="0" >
        <wlevs:instance-property name="eventPartitioner" value="true"
/>
        <wlevs:listener ref="processor1" />
    </wlevs:channel>
...
```

```

        <wlevs:listener ref="processor2" />
        <wlevs:listener ref="processor3" />
        <wlevs:source ref="inbound" />
    </wlevs:channel>
    ...

```

- Edit the Oracle Event Processing server `config.xml` file to add a `work-manager` element.

If this work manager is shared by more than one component (such as other adapters and Jetty), then set the `min-threads-constraint` and `max-threads-constraint` elements each to a value greater than the number of listeners.

If this work manager is not shared by more than one component (that is, it is dedicated to the upstream adapter in this configuration), then set the `min-threads-constraint` and `max-threads-constraint` elements equal to the number of listeners.

```

    ...
    <work-manager>
        <name>adapterWorkManager</name>
        <min-threads-constraint>3</min-threads-constraint>
        <max-threads-constraint>3</max-threads-constraint>
    </work-manager>
    ...

```

For more information, see [Section , "work-manager"](#).

- Edit the component configuration file to configure the upstream adapter with this `work-manager`.

```

    ...
    <adapter>
        <name>inbound</name>
        <work-manager-name>adapterWorkManager</work-manager-name>
        ...
    </adapter>
    ...

```

## 12. Assemble and deploy your application.

For more information, see [Chapter 23, "Assembling and Deploying Oracle Event Processing Applications"](#).

At runtime, the channel uses your `EventPartitioner` to determine how to dispatch each incoming event to its listeners.

## Configuring Scalability With the `ActiveActiveGroupBean`

This section describes how to configure your Oracle Event Processing application to use the `ActiveActiveGroupBean` to partition an incoming JMS event stream by selector, including:

- [Section , "How to Configure Scalability in a JMS Application Without Oracle Event Processing High Availability"](#)
- [Section , "How to Configure Scalability in a JMS Application With Oracle Event Processing High Availability"](#)
- [Section , "How to Configure the `ActiveActiveGroupBean` Group Pattern Match"](#)

For more information, see [Section , "ActiveActiveGroupBean"](#).

## How to Configure Scalability in a JMS Application Without Oracle Event Processing High Availability

You can use the `ActiveActiveGroupBean` to partition an incoming JMS event stream by selector in a multi-server domain for an application that does not use Oracle Event Processing high availability.

For information on how to use the `ActiveActiveGroupBean` in an Oracle Event Processing high availability application, see [Section , "How to Configure Scalability in a JMS Application With Oracle Event Processing High Availability"](#).

For more information, see [Section , "Scalability in an Oracle Event Processing Application Using the ActiveActiveGroupBean Without High Availability"](#).

### To configure scalability in a JMS application without Oracle Event Processing high availability:

1. Create a multi-server domain.

For more information, see "Introduction to Multi-Server Domains" in the *Oracle Fusion Middleware Administrator's Guide for Oracle Event Processing*.

In this example, the deployment group is named `MyDeploymentGroup`.

2. Configure the Oracle Event Processing server configuration file on each Oracle Event Processing server to add the appropriate `ActiveActiveGroupBean` notification group to the `groups` child element of the `cluster` element.

The Oracle Event Processing 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.

For example, [Table 25–4](#) shows `cluster` elements for Oracle Event Processing servers `ocep-server-1`, `ocep-server-2`, `ocep-server-3`, and `ocep-server-4`. The deployment group is `MyDeploymentGroup` and notification groups are defined using default `ActiveActiveGroupBean` notification group naming.

Optionally, you can specify your own group naming convention as [Section , "How to Configure the ActiveActiveGroupBean Group Pattern Match"](#) describes.

**Table 25–3 Oracle Event Processing Server Configuration File groups Element Configuration**

Partition	cluster Element
ocep-server-1	<pre>&lt;cluster&gt;   &lt;server-name&gt;ocep-server-1&lt;/server-name&gt;   ...   &lt;enabled&gt;coherence&lt;/enabled&gt;   ...   &lt;groups&gt;MyDeploymentGroup, ActiveActiveGroupBean_group1&lt;/groups&gt; &lt;/cluster&gt;</pre>
ocep-server-2	<pre>&lt;cluster&gt;   &lt;server-name&gt;ocep-server-2&lt;/server-name&gt;   ...   &lt;enabled&gt;coherence&lt;/enabled&gt;   ...   &lt;groups&gt;MyDeploymentGroup, ActiveActiveGroupBean_group2&lt;/groups&gt; &lt;/cluster&gt;</pre>

**Table 25–3 (Cont.) Oracle Event Processing Server Configuration File groups Element Configuration**

Partition	cluster Element
ocep-server-3	<pre>&lt;cluster&gt;   &lt;server-name&gt;ocep-server-3&lt;/server-name&gt;   ...   &lt;enabled&gt;coherence&lt;/enabled&gt;   ...   &lt;groups&gt;MyDeploymentGroup, ActiveActiveGroupBean_group3&lt;/groups&gt; &lt;/cluster&gt;</pre>
ocep-server-4	<pre>&lt;cluster&gt;   &lt;server-name&gt;ocep-server-4&lt;/server-name&gt;   ...   &lt;enabled&gt;coherence&lt;/enabled&gt;   ...   &lt;groups&gt;MyDeploymentGroup, ActiveActiveGroupBean_group4&lt;/groups&gt; &lt;/cluster&gt;</pre>

3. Create an Oracle Event Processing application.
4. Configure the EPN assembly file to add an ActiveActiveGroupBean element as [Example 25–10](#) shows.

**Example 25–6 ActiveActiveGroupBean bean Element**

```
<bean id="clusterAdapter" class="com.oracle.cep.cluster.hagroups.ActiveActiveGroupBean">
</bean>
```

5. Define a parameterized message-selector in the jms-adapter element for the JMS inbound adapters.

Edit the component configuration file to add group-binding child elements to the jms-adapter element for the JMS inbound adapters.

Add one group-binding element for each possible JMS message-selector value as [Example 25–12](#) shows.

**Example 25–7 jms-adapter Selector Definition for ocep-server-1**

```
<jms-adapter>
  <name>JMSInboundAdapter</name>
  <event-type>StockTick</event-type>
  <jndi-provider-url>t3://ppurich-pc:7001</jndi-provider-url>
  <destination-jndi-name>./Topic1</destination-jndi-name>
  <user>weblogic</user>
  <password>weblogic1</password>
  <work-manager>JettyWorkManager</work-manager>
  <concurrent-consumers>1</concurrent-consumers>
  <session-transacted>>true</session-transacted>
  <message-selector>${CONDITION}</message-selector>
  <bindings>
    <group-binding group-id="ActiveActiveGroupBean_group1">
      <param id="CONDITION">acctid > 400</param>
    </group-binding>
    <group-binding group-id="ActiveActiveGroupBean_group2">
      <param id="CONDITION">acctid BETWEEN 301 AND 400</param>
    </group-binding>
    <group-binding group-id="ActiveActiveGroupBean_group3">
      <param id="CONDITION">acctid BETWEEN 201 AND 300</param>
    </group-binding>
    <group-binding group-id="ActiveActiveGroupBean_group4">
      <param id="CONDITION">acctid <= 200</param>
    </group-binding>
  </bindings>
```

```
</bindings>
</jms-adapter>
```

In this configuration, when the application is deployed to an Oracle Event Processing server with a `cluster` element groups child element that contains `ActiveActiveGroupBean_group1`, then the `CONDITION` parameter is defined as `acctid > 400` and the application processes events whose `acctid` property is greater than 400.

---

**Note:** Each in-bound JMS adapter must listen to a different topic.

For more information, see [Chapter 11, "Integrating the Java Message Service"](#).

---

6. Deploy your application to the deployment group of your multi-server domain.

For more information, see [Chapter 23, "Assembling and Deploying Oracle Event Processing Applications"](#).

At runtime, each Oracle Event Processing server configures its instance of the application with the `message-selector` that corresponds to its `ActiveActiveGroupBean` notification group. This partitions the JMS topic so that each instance of the application processes a subset of the total number of messages in parallel.

## How to Configure Scalability in a JMS Application With Oracle Event Processing High Availability

You can use the `ActiveActiveGroupBean` to partition an incoming JMS event stream in a multi-server domain with Oracle Event Processing high availability.

This procedure uses the example application from [Section , "How to Configure Precise Recovery With JMS"](#), including the example EPN that [Figure 25–6](#) shows, the corresponding EPN assembly file that [Example 25–8](#) shows, and the corresponding component configuration file that [Example 25–9](#) shows.

**Figure 25–6** *Precise Recovery With JMS EPN*



### Example 25–8 Precise Recovery With JMS EPN Assembly File

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

  <wlevs:event-type-repository>
    <wlevs:event-type type-name="StockTick">
      <wlevs:properties>
        <wlevs:property name="lastPrice" type="double" />
        <wlevs:property name="symbol" type="char" />
      </wlevs:properties>
    </wlevs:event-type>
  </wlevs:event-type-repository>

  <wlevs:adapter id="JMSInboundAdapter" provider="jms-inbound">
```

```

        <wlevs:listener ref="myHaInputAdapter" />
    </wlevs:adapter>

    <wlevs:adapter id="myHaInputAdapter" provider="ha-inbound" >
        <wlevs:instance-property name="keyProperties" value="sequenceNo" />
        <wlevs:instance-property name="timeProperty" value="inboundTime" />
    </wlevs:adapter>

    <wlevs:channel id="channel1" event-type="StockTick">
        <wlevs:listener ref="processor1" />
        <wlevs:source ref="myHaInputAdapter" />
        <wlevs:application-timestamped>
            <wlevs:expression>inboundTime</wlevs:expression>
        </wlevs:application-timestamped>
    </wlevs:channel>

    <wlevs:processor id="processor1">
        <wlevs:listener ref="channel2" />
    </wlevs:processor>

    <wlevs:channel id="channel2" event-type="StockTick">
        <wlevs:listener ref="myHaCorrelatingAdapter" />
    </wlevs:channel>

    <wlevs:adapter id="myHaCorrelatingAdapter" provider="ha-correlating" >
        <wlevs:instance-property name="correlatedSource" ref="clusterCorrelatingOutstream" />
        <wlevs:instance-property name="failOverDelay" value="2000" />
        <wlevs:listener ref="JMSOutboundAdapter" />
    </wlevs:adapter>

    <wlevs:adapter id="JMSOutboundAdapter" provider="jms-outbound">
    </wlevs:adapter>

    <wlevs:adapter id="JMSInboundAdapter2" provider="jms-inbound">
    </wlevs:adapter>

    <wlevs:channel id="clusterCorrelatingOutstream" event-type="StockTick" advertise="true">
        <wlevs:source ref="JMSInboundAdapter2" />
    </wlevs:channel>
</beans>

```

### Example 25–9 Precise Recovery With JMS Component Configuration Assembly File

```

<?xml version="1.0" encoding="UTF-8"?>
<wlevs:config
    xmlns:wlevs="http://www.bea.com/ns/wlevs/config/application"
    xmlns:ha="http://www.oracle.com/ns/cep/config/cluster">
    <processor>
        <name>processor1</name>
        <rules>
            <query id="helloworldRule">
                <![CDATA[ select * from channel1 [Now] ]]>
            </query>
        </rules>
    </processor>
    <jms-adapter>
        <name>JMSInboundAdapter</name>
        <event-type>StockTick</event-type>
        <jndi-provider-url>t3://ppurich-pc:7001</jndi-provider-url>
        <destination-jndi-name>./Topic1</destination-jndi-name>
        <session-transacted>true</session-transacted>
        ...
    </jms-adapter>
    <jms-adapter>
        <name>JMSInboundAdapter2</name>

```



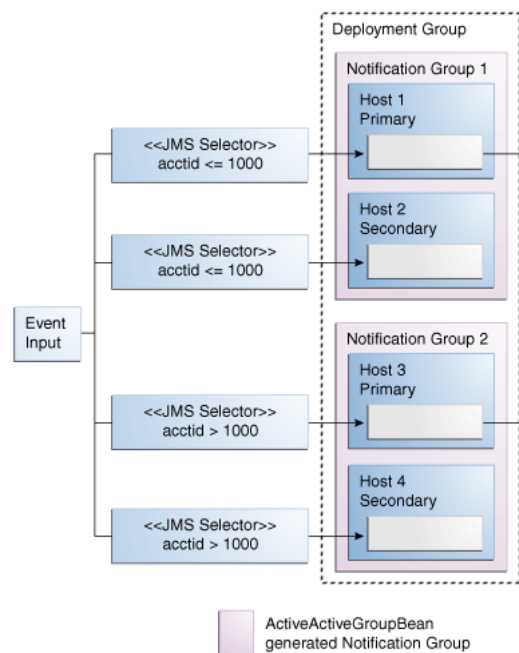
```

<event-type>StockTick</event-type>
<jndi-provider-url>t3://ppurich-pc:7001</jndi-provider-url>
<destination-jndi-name>./Topic2</destination-jndi-name>
<session-transacted>>true</session-transacted>
...
</jms-adapter>
<jms-adapter>
  <name>JMSOutboundAdapter</name>
  <event-type>StockTick</event-type>
  <jndi-provider-url>t3://ppurich-pc:7001</jndi-provider-url>
  <destination-jndi-name>./Topic2</destination-jndi-name>
  <session-transacted>>true</session-transacted>
...
</jms-adapter>
</wlevs:config>

```

This procedure will create the Oracle Event Processing high availability configuration that [Figure 25–7](#) shows.

**Figure 25–7 Oracle Event Processing ActiveActiveGroupBean With High Availability**



For more information, see [Section , "Scalability in an Oracle Event Processing Application Using the ActiveActiveGroupBean With High Availability"](#).

**To configure scalability in a JMS application with Oracle Event Processing high availability:**

1. Create a multi-server domain.

For more information, see "Introduction to Multi-Server Domains" in the *Oracle Fusion Middleware Administrator's Guide for Oracle Event Processing*.

In this example, the deployment group is named `MyDeploymentGroup`.

2. Configure the Oracle Event Processing server configuration file on each Oracle Event Processing server to add the appropriate `ActiveActiveGroupBean` notification group to the `groups` child element of the `cluster` element.

The Oracle Event Processing 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.

For example, [Table 25–4](#) shows `cluster` elements for Oracle Event Processing servers `ocep-server-1`, `ocep-server-2`, `ocep-server-3`, and `ocep-server-4`. The deployment group is `MyDeploymentGroup` and notification groups are defined using default `ActiveActiveGroupBean` notification group names.

Note that `ocep-server-1` and `ocep-server-2` use the same notification group name (`ActiveActiveGroupBean_group1`) and `ocep-server-3` and `ocep-server-4` use the same notification group name (`ActiveActiveGroupBean_group2`).

Optionally, you can specify your own group naming convention as [Section , "How to Configure the ActiveActiveGroupBean Group Pattern Match"](#) describes.

**Table 25–4 Oracle Event Processing Server Configuration File groups Element Configuration**

Partition	cluster Element
ocep-server-1	<pre>&lt;cluster&gt;   &lt;server-name&gt;ocep-server-1&lt;/server-name&gt;   ...   &lt;enabled&gt;coherence&lt;/enabled&gt;   ...   &lt;groups&gt;MyDeploymentGroup, ActiveActiveGroupBean_group1&lt;/groups&gt; &lt;/cluster&gt;</pre>
ocep-server-2	<pre>&lt;cluster&gt;   &lt;server-name&gt;ocep-server-2&lt;/server-name&gt;   ...   &lt;enabled&gt;coherence&lt;/enabled&gt;   ...   &lt;groups&gt;MyDeploymentGroup, ActiveActiveGroupBean_group1&lt;/groups&gt; &lt;/cluster&gt;</pre>
ocep-server-3	<pre>&lt;cluster&gt;   &lt;server-name&gt;ocep-server-3&lt;/server-name&gt;   ...   &lt;enabled&gt;coherence&lt;/enabled&gt;   ...   &lt;groups&gt;MyDeploymentGroup, ActiveActiveGroupBean_group2&lt;/groups&gt; &lt;/cluster&gt;</pre>
ocep-server-4	<pre>&lt;cluster&gt;   &lt;server-name&gt;ocep-server-4&lt;/server-name&gt;   ...   &lt;enabled&gt;coherence&lt;/enabled&gt;   ...   &lt;groups&gt;MyDeploymentGroup, ActiveActiveGroupBean_group2&lt;/groups&gt; &lt;/cluster&gt;</pre>

3. Create an Oracle Event Processing high availability application.

For more information, see [Chapter 24, "Developing Applications for High Availability"](#).

4. Configure the EPN assembly file to add an `ActiveActiveGroupBean` element as [Example 25–10](#) shows.

**Example 25–10 `ActiveActiveGroupBean` bean Element**

```
<bean id="clusterAdapter" class="com.oracle.cep.cluster.hagroups.ActiveActiveGroupBean">
</bean>
```

5. Edit the component configuration file to configure a `jms-adapter` element for the inbound JMS adapters as [Example 25–11](#) shows:
  - Each in-bound JMS adapter must listen to a different topic.
  - Set `session-transacted` to true.

**Example 25–11 `jms-adapter` Element for Inbound JMS Adapters**

```
<?xml version="1.0" encoding="UTF-8"?>
<wlevs:config
  xmlns:wlevs="http://www.bea.com/ns/wlevs/config/application"
  xmlns:ha="http://www.oracle.com/ns/cep/config/cluster">
  ...
  <jms-adapter>
    <name>JMSInboundAdapter</name>
    <event-type>StockTick</event-type>
    <jndi-provider-url>t3://ppurich-pc:7001</jndi-provider-url>
    <destination-jndi-name>./Topic1</destination-jndi-name>
    <session-transacted>true</session-transacted>
  ... </jms-adapter>
  <jms-adapter>
    <name>JMSInboundAdapter2</name>
    <event-type>StockTick</event-type>
    <jndi-provider-url>t3://ppurich-pc:7001</jndi-provider-url>
    <destination-jndi-name>./Topic2</destination-jndi-name>
    <session-transacted>true</session-transacted>
  ... </jms-adapter>
</wlevs:config>
```

For more information, see [Chapter 11, "Integrating the Java Message Service"](#).

6. Define a parameterized `message-selector` in the `jms-adapter` element for each JMS inbound adapter.

Edit the component configuration file to add `group-binding` child elements to the `jms-adapter` element for the JMS inbound adapters.

Add one `group-binding` element for each possible JMS `message-selector` value as [Example 25–12](#) shows.

**Example 25–12 `jms-adapter` Selector Definition for `ocep-server-1`**

```
<jms-adapter>
  <name>JMSInboundAdapter</name>
  <event-type>StockTick</event-type>
  <jndi-provider-url>t3://ppurich-pc:7001</jndi-provider-url>
  <destination-jndi-name>./Topic1</destination-jndi-name>
  <session-transacted>true</session-transacted>
  <message-selector>${CONDITION}</message-selector>
  <bindings>
    <group-binding group-id="ActiveActiveGroupBean_group1">
      <param id="CONDITION">acctid <= 1000</param>
    </group-binding>
    <group-binding group-id="ActiveActiveGroupBean_group2">
      <param id="CONDITION">acctid > 1000</param>
    </group-binding>
  </bindings>
```

```

    </bindings>
</jms-adapter>

```

In this configuration, when the application is deployed to an Oracle Event Processing server with a `cluster` element `groups` child element that contains `ActiveActiveGroupBean_group1`, then the `CONDITION` parameter is defined as `acctid <= 1000` and the application processes events whose `acctid` property is less than or equal to 1000. Similarly, when the application is deployed to an Oracle Event Processing server with a `cluster` element `groups` child element that contains `ActiveActiveGroupBean_group2`, then the `CONDITION` parameter is defined as `acctid > 1000` and the application processes events whose `acctid` property is greater than 1000.

7. Edit the component configuration file to configure a `jms-adapter` element for the outbound JMS adapter as [Example 25–13](#) shows:
  - Configure the out-bound JMS adapter with the same topic as the correlating in-bound adapter (in this example, `JMSInboundAdapter2: ./Topic2`).
  - Set `session-transacted` to `true`.

#### **Example 25–13** *jms-adapter Element for Outbound JMS Adapters*

```

<?xml version="1.0" encoding="UTF-8"?>
<wlevs:config
  xmlns:wlevs="http://www.bea.com/ns/wlevs/config/application"
  xmlns:ha="http://www.oracle.com/ns/cep/config/cluster">
  ...
  <jms-adapter>
    <name>JMSInboundAdapter</name>
    <event-type>StockTick</event-type>
    <jndi-provider-url>t3://ppurich-pc:7001</jndi-provider-url>
    <destination-jndi-name>./Topic1</destination-jndi-name>
    <session-transacted>true</session-transacted>
  ...   </jms-adapter>
  <jms-adapter>
    <name>JMSInboundAdapter2</name>
    <event-type>StockTick</event-type>
    <jndi-provider-url>t3://ppurich-pc:7001</jndi-provider-url>
    <destination-jndi-name>./Topic2</destination-jndi-name>
    <session-transacted>true</session-transacted>
  ...   </jms-adapter>
  <jms-adapter>
    <name>JMSOutboundAdapter</name>
    <event-type>StockTick</event-type>
    <jndi-provider-url>t3://ppurich-pc:7001</jndi-provider-url>
    <destination-jndi-name>./Topic2</destination-jndi-name>
    <session-transacted>true</session-transacted>
  ...   </jms-adapter>
</wlevs:config>

```

For more information, see [Chapter 11, "Integrating the Java Message Service"](#).

8. Deploy your application to the deployment group of your multi-server domain.

For more information, see [Chapter 23, "Assembling and Deploying Oracle Event Processing Applications"](#).

At runtime, each Oracle Event Processing server configures its instance of the application with the `message-selector` that corresponds to its `ActiveActiveGroupBean` notification group. This partitions the JMS topic so that each instance of the application processes a subset of the total number of messages in parallel.

If the active Oracle Event Processing server in an `ActiveActiveGroupBean` group goes down, the Oracle Event Processing server performs an Oracle Event Processing high availability failover to the standby Oracle Event Processing server in that `ActiveActiveGroupBean` group.

### How to Configure the `ActiveActiveGroupBean` Group Pattern Match

By default, the `ActiveActiveGroupBean` creates notification groups named:

```
ActiveActiveGroupBean_X
```

Where *X* is a string.

At runtime, the `ActiveActiveGroupBean` scans the existing groups defined on the Oracle Event Processing server and applies a default pattern match of:

```
ActiveActiveGroupBean_\\w+
```

When it finds a match, it creates a notification group of that name.

Optionally, you can define your own group pattern to specify a different notification group naming pattern.

### How to configure the `ActiveActiveGroupBean` group pattern match:

1. Configure the EPN assembly file to add a `groupPattern` attribute to your `ActiveActiveGroupBean` element as [Example 25–14](#) shows.

#### **Example 25–14** *ActiveActiveGroupBean bean Element With groupPattern Attribute*

```
<bean id="clusterAdapter" class="com.oracle.cep.cluster.hagroups.ActiveActiveGroupBean">  
  <property name="groupPattern" value="MyNotificationGroupPattern*" />  
</bean>
```

2. Specify a value for the `groupPattern` attribute that matches the cluster group naming convention you want to use for notification groups.



---

---

## Extending Component Configuration

This chapter describes how to use an XML schema to extend the default configuration for adapters and event beans used with Oracle Event Processing, including information on using Java annotations to generate the configuration XML.

This chapter includes the following sections:

- [Overview of Extending Component Configuration](#)
- [Extending Component Configuration](#)
- [Programming Access to the Configuration of a Custom Adapter or Event Bean](#)

### Overview of Extending Component Configuration

Adapters and event beans have default configuration data. This default configuration is typically adequate for simple and basic applications.

However, you can also extend this configuration by using a XML Schema Definition (XSD) schema to specify a *new* XML format of an adapter configuration file that extends the built-in XML type provided by Oracle Event Processing. 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 and Java Architecture for XML Binding (JAXB).

You can extend component configuration by:

- Annotating your adapter or event bean Java class with the annotations that `javax.xml.bind.annotation` specifies.  
See [Section , "Extending Component Configuration Using Annotations"](#).
- Manually generating an XSD.  
See [Section , "Extending Component Configuration Using an XSD"](#).
- Manually generating a custom schema which does not extend the application schema.

This allows you to create custom configuration in your own namespace without having to define all the other elements. This mechanism functions like the annotation approach after you generate the schema.

For more information, see:

- [Chapter 15, "Integrating an External Component Using a Custom Adapter"](#)
- [Section , "Configuring a Java Class as an Event Bean"](#)

- [Section , "Component Configuration Schema wlevs\\_application\\_config.xsd"](#)

## Extending Component Configuration Using Annotations

The simplest and most efficient way to extend component configuration is to annotate your adapter or event bean Java class using the annotations that `javax.xml.bind.annotation` specifies.

Oracle Event Processing supports the inclusion of multiple namespaces in an EPN configuration file as well as supporting sourcing configuration information from the providing bundle rather than the client bundle. Oracle Event Processing scans for multiple `ObjectFactories` in the accessible class-space and each of these will be initialized through `createConfig`.

The schema search takes into account the `wlevs:factory element provider-schema child element` in adapter bundles. So if you are defining an adapter in its own bundle you can put the schema in that bundle as long as you define the `provider-bundle` property.

Oracle recommends that you use `elementFormDefault="unqualified"` so that locally defined elements will not have a namespace, but global elements will leverage the `targetNamespace`. This will avoid name clashes across schemas without requiring excessive prefixing.

For more information, see <http://www.xfront.com/HideVersusExpose.html>.

For more information, see:

- [Section , "How to Extend Component Configuration Using Annotations"](#)
- <http://java.sun.com/javaee/6/docs/api/javax/xml/bind/annotation/package-summary.html>

## Extending Component Configuration Using an XSD

If you require more detailed control over your custom component configuration, you can extend your component configuration by creating your own XSD.

For more information, see:

- [Section , "How to Extend Component Configuration Using an XSD"](#)
- <http://jaxb.java.net/>

## Extending Component Configuration

You can extend component configuration in either of the following ways:

For more information, see [Section , "Overview of Extending Component Configuration"](#).

### How to Extend Component Configuration Using Annotations

The simplest and most efficient way to extend component configuration is to annotate your adapter or event bean Java class.

Alternatively, you can extend component configuration by creating your own XSD as [Section , "How to Extend Component Configuration Using an XSD"](#) describes.

For more information, see [Section , "Extending Component Configuration Using Annotations"](#).



**To extend component configuration using annotations:**

1. Implement your custom adapter or event bean Java class.

For more information, see:

- [Section , "Implementing a Custom Adapter"](#)
- [Chapter 16, "Handling Events with Java"](#)

2. Annotate the attributes of your custom adapter or event bean to specify the component configuration using the annotations that `javax.xml.bind.annotation` specifies.

Important `javax.xml.bind.annotation` annotations include:

- `@XmlElement`: property is an optional part of the component configuration.
- `@XmlElement(required=true)`: property is a required part of the component configuration.
- `@XmlTransient`: property is not part of the component configuration.
- `@XmlJavaTypeAdapter`: property elements annotated with this can specify custom handling to accommodate most Java data types.

---

**Note:** If you require extensive use of `@XmlJavaTypeAdapter`, consider defining your own custom schema as [Section , "How to Extend Component Configuration Using an XSD"](#) describes.

---



---

**Note:** A property without an annotation is assumed to be an optional configuration property (default: `@XmlElement`).

---

[Example 26–1](#) shows a custom adapter implementation annotated with `javax.xml.bind.annotation` annotations to specify:

- `count`: not part of the component configuration.
- `doit`: required part of the component configuration.
- `size`: required part of the component configuration; maps to instance property `howBig`.

**Example 26–1 Annotated Custom Adapter Implementation**

```
@XmlType(name="SampleAdapterConfig", namespace="http://www.oracle.com/ns/cep/config/sample")
public class SampleAdapterImpl implements Adapter {
    @XmlTransient
    private int count;

    @XmlElement(name="size")
    private int howBig;

    @XmlElement(required=true)
    private boolean doit;

    ...

    public void setDoit(boolean doit) {
        this.doit = doit;
    }

    public boolean getDoit() {
```

```

        return doit;
    }
}

```

3. Within your custom adapter or event bean code, access the extended configuration as [Section , "Programming Access to the Configuration of a Custom Adapter or Event Bean"](#) describes.
4. Modify the component configuration XML file that describes the custom components of your application.

For more information, see [Section , "Configuring a Custom Adapter in a Component Configuration File"](#).

5. When you create the component configuration XML file that describes the components of your application, be sure to use the extended XSD file as its description. In addition, be sure you identify the namespace for this schema rather than the default schema.

[Example 26–2](#) shows a component configuration file for the custom adapter in [Example 26–1](#).

**Example 26–2 Extended Component Configuration: Annotations**

```

<?xml version="1.0" encoding="UTF-8"?>
<app:config
  xmlns:app="http://www.bea.com/ns/wlevs/config/application"
  xmlns:sample="http://www.oracle.com/ns/cep/config/sample"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="
    http://www.bea.com/ns/wlevs/config/application
    http://www.bea.com/ns/wlevs/config/application/wlevs_application_config.xsd
    http://www.oracle.com/ns/cep/config/sample
    http://www.oracle.com/ns/cep/config/sample/ocep_sample_config.xsd">
  <processor>
    <name>clusterProcessor</name>
    <rules>
      <query id="clusterRule"><![CDATA[ select * from clusterInstream [Now] ]]></query>
    </rules>
  </processor>
  <sample:adapter>
    <name>myadapter</name>
    <config>
      <size>15</size>      <!-- optional -->
      <doit>true</doit>    <!-- required -->
    </config>
  </sample:adapter>
</app:config>

```

---



---

**Note:** The extended component configuration schema requires a nested config element as [Example 26–5](#) shows.

---



---

6. Package and deploy your application.

For more information, see [Chapter 23, "Assembling and Deploying Oracle Event Processing Applications"](#).

## How to Extend Component Configuration Using an XSD

You can extend the component configuration of a custom adapter or event bean using your own XSD.

Alternatively, you can extend component configuration by annotating your adapter or event bean Java class as [Section , "How to Extend Component Configuration Using Annotations"](#) describes.

For more information, see [Section , "Extending Component Configuration Using an XSD"](#).

### To extend component configuration using an XSD:

1. Create the new XSD Schema file that describes the extended adapter or event bean 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 Event Processing, to describe them.

See [Section , "Creating the XSD Schema File"](#).

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 from Sun's GlassFish reference implementation. This Ant task is included in the Oracle Event Processing 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="${base.dir}/../../../../../../modules" />
<property name="wlrllib.dir" value="${base.dir}/../../../../../../modules"/>
<path id="classpath">
  <pathelement location="${output.dir}" />
  <fileset dir="${sharedlib.dir}">
    <include name="*.jar" />
  </fileset>
  <fileset dir="${wlrllib.dir}">
    <include name="*.jar"/>
  </fileset>
</path>
<taskdef name="xjc" classname="com.sun.tools.xjc.XJCTask">
  <classpath refid="classpath" />
</taskdef>
<target name="generate" depends="clean, init">
  <copy file="../../../../xsd/wlevs_base_config.xsd"
    todir="src/main/resources/extension" />
  <copy file="../../../../xsd/wlevs_application_config.xsd"
    todir="src/main/resources/extension" />
  <xjc extension="true" destdir="${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 Event Processing 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 Event Processing XSD files.

For more information, see

<http://jaxb.java.net/nonav/2.0.2/docs/xjcTask.html>.

3. Compile these generated Java files into classes.
4. Package the compiled Java class files in your application bundle.  
See [Section , "Assembling an Oracle Event Processing Application"](#).
5. Program your custom adapter or event bean.  
For more information, see:
  - [Section , "Implementing a Custom Adapter."](#)
  - [Chapter 16, "Handling Events with Java"](#)
6. Within your custom adapter or event bean code, access the extended configuration as [Section , "Programming Access to the Configuration of a Custom Adapter or Event Bean"](#) describes.
7. When you create the component configuration XML file that describes the components of your application, be sure to use the extended XSD file as its description. In addition, be sure you identify the namespace for this schema rather than the default schema.

[Example 26–3](#) shows a component configuration file for the XSD you created in [Section , "Creating the XSD Schema File"](#).

**Example 26–3 Extended Component Configuration File: XSD**

```
<?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>
```

### Creating the XSD Schema File

The new XSD schema file extends the `wlevs_application_config.xsd` XSD schema 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 , "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.

For more information, see [Section , "Component Configuration Schema `wlevs\_application\_config.xsd`"](#).

**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 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/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:**

```

<xs:import
  namespace="http://www.bea.com/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 them 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—For a description of the default processor configuration, see:
  - [Section , "Overview of Oracle CQL Processor Configuration"](#)
  - [Section , "Overview of EPL Processor Component Configuration"](#)

- `DefaultStreamConfig`—For a description of the default channel configuration, see [Section , "Overview of 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>
```

- 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>
```

**Complete Example of an Extended XSD Schema File** Use the following extended XSD file as a template:

```
<?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/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/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:complexType>
</xs:schema>
```

## Programming Access to the Configuration of a Custom 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 Event Processing 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 <http://jaxb.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 Event Processing 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.

You can access this component configuration at runtime in either of the following ways:

- Using resource injection as [Section , "How to Access Component Configuration Using Resource Injection"](#) describes.

This is the simplest and most efficient way to access component configuration at runtime.

- Using callbacks as [Section , "How to Access Component Configuration Using Lifecycle Callbacks"](#).

This is the most flexible way to access component configuration at runtime. Consider this option if you cannot easily accomplish your intentions using resource injection.

---



---

**Note:** Clients needing to use schema from an adapter provider must import the appropriate package from the provider bundle so that the provider's `ObjectFactory` is visible to the client bundle.

---



---

### How to Access Component Configuration Using Resource Injection

By default, Oracle Event Processing configures adapters by direct injection of their Java bean properties followed by the usual configuration callbacks.

Consider the annotated custom adapter implementation that [Example 26–4](#) shows.

#### **Example 26–4 Custom Adapter Implementation**

```
@XmlType(name="SampleAdapterConfig", namespace="http://www.oracle.com/ns/cep/config/sample")
public class SampleAdapterImpl implements Adapter {
    private boolean doit;

    public void setDoit(boolean doit) {
        this.doit = doit;
    }

    public boolean getDoit() {
        return doit;
    }
}
```

And consider the component configuration file for an instance of this custom adapter that [Example 26–5](#)

**Example 26–5 Extended Component Configuration**

```
<?xml version="1.0" encoding="UTF-8"?>
<app:config
  xmlns:app="http://www.bea.com/ns/wlevs/config/application"
  xmlns:sample="http://www.oracle.com/ns/cep/config/sample"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="
    http://www.bea.com/ns/wlevs/config/application
    http://www.bea.com/ns/wlevs/config/application/wlevs_application_config.xsd
    http://www.oracle.com/ns/cep/config/sample
    http://www.oracle.com/ns/cep/config/sample/ocep_sample_config.xsd">
  <processor>
    <name>clusterProcessor</name>
    <rules>
      <query id="clusterRule"><![CDATA[ select * from clusterInstream [Now] ]]></query>
    </rules>
  </processor>
  <sample:adapter>
    <name>myadapter</name>
    <config>
      <doit>true</doit>
    </config>
  </sample:adapter>
</app:config>
```

At runtime, by default Oracle Event Processing directly injects the value (`true`) of the Java bean property `doit`.

---

**Note:** The extended component configuration schema requires a nested `config` element as [Example 26–5](#) shows.

---

For more information, see:

- [Section , "How to Access Component Configuration Using Lifecycle Callbacks"](#)
- [Section , "Configuring Oracle Event Processing Resource Access"](#)

## How to Access Component Configuration Using Lifecycle Callbacks

In your adapter implementation, you can use metadata annotations to specify the Java methods that are invoked by Oracle Event Processing at runtime.

Oracle Event Processing passes an instance of the configuration Java class to these 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 data 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`.

This section describes:

- [Section , "Lifecycle Callback Annotations"](#)
- [Section , "Lifecycle"](#)

### Lifecycle Callback Annotations

You can use the following metadata annotations to specify various lifecycle callback methods:

- `com.bea.wlevs.management.Activate`—Specifies the method invoked when the configuration is activated.

See [Section , "com.bea.wlevs.configuration.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 , "com.bea.wlevs.configuration.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 , "com.bea.wlevs.configuration.Rollback"](#) for additional details about using this annotation in your adapter code.

For more information, see [Section , "Lifecycle"](#).

### Lifecycle

Oracle Event Processing follows the following lifecycle during custom adapter and event bean instantiation:

1. Create adapter or event bean instance.
2. Inject static properties.
3. Call `afterPropertiesSet`.
4. Prepare phase:
  - a. If `@Prepare` with one or more configuration arguments is present, call it.
  - b. Otherwise, directly inject configuration properties.  
See [Section , "How to Access Component Configuration Using Resource Injection"](#).
  - c. If `@Prepare` without arguments is present, call it.
5. Activate phase:
  - a. If `@Activate` with one or more configuration arguments is present, call it.
  - b. If `@Activate` without arguments is present, call it.
6. Call `afterConfigurationActive`.
7. Continue with other configuration.



---

---

## Performance Tuning

This chapter describes techniques for improving Oracle Event Processing application performance by using partitioning and batching, and includes information specific to high availability performance tuning.

This chapter includes the following sections:

- [EPN Performance Tuning](#)
- [High Availability Performance Tuning](#)

### EPN Performance Tuning

When creating your EPN, consider the following performance tuning options:

- [Section , "Event Partitioner Channel"](#)
- [Section , "Batching Channel"](#)
- [Section , "Scalability Using the ActiveActiveGroupBean"](#)

For more information, see [Chapter 25, "Developing Scalable Applications"](#).

### Event Partitioner Channel

You can improve scalability by configuring an event partitioner channel. When you configure a channel to use an event partitioner, each time an incoming event arrives, the channel selects a listener and dispatches the event to that listener instead of broadcasting each event to every listener for partitioning events on a channel across its output event sinks.

For more information, see [Section , "EventPartitioner"](#).

### Batching Channel

By default, a channel processes events as they arrive. Alternatively, you can configure a channel to batch events together that have the same timestamp by setting the `wlevs:channel` attribute `batching` to `true`.

For more information, see [Section , "Batch Processing Channels"](#).

### Scalability Using the ActiveActiveGroupBean

Using the `com.oracle.cep.cluster.hagroups.ActiveActiveGroupBean`, you can partition an incoming JMS stream in Oracle Event Processing applications by utilizing the notification groups that the `ActiveActiveGroupBean` creates.

For more information, see [Section , "ActiveActiveGroupBean"](#).

## High Availability Performance Tuning

When creating high-availability applications for deployment to multi-server domains, consider the following performance tuning options:

- [Section , "Host Configuration"](#)
- [Section , "High Availability Input Adapter and Quality of Service"](#)
- [Section , "High Availability Input Adapter Configuration"](#)
- [Section , "Broadcast Output Adapter Configuration"](#)
- [Section , "Oracle Coherence Performance Tuning Options"](#)

For more information, see [Section , "Designing an Oracle Event Processing Application for High Availability"](#)

### Host Configuration

If you only want availability and are not concerned with recovery time, then it is possible to use smaller, less equipped hosts as secondaries. However, to maximize high availability performance, ensure that all hosts in the multi-server domain are identical: same number and type of processor, same quantity of memory, same number and size of disks.

### High Availability Input Adapter and Quality of Service

The Oracle Event Processing high availability input adapter is applicable to all high availability quality of service options. However, because the high availability input adapter increases performance overhead, it is not appropriate for some high availability quality of service options (such as [Section , "Simple Failover"](#) and [Section , "Simple Failover with Buffering"](#)).

If you are using application time from the event then you do not need to use the input adapter. Application time from the event is always preferable from a performance standpoint.

### High Availability Input Adapter Configuration

Consider increasing the `batch-size` to reduce the amount of time the primary server spends broadcasting event messages and to reduce the amount of time the secondary servers spend processing these messages.

Increasing the `batch-size` may increase the likelihood of missed and duplicate events if the primary fails before broadcasting an event message with a large number of events.

For more information, see [Table 24-5, "Child Elements of ha-inbound-adapter for the High Availability Input Adapter"](#).

### Broadcast Output Adapter Configuration

Consider decreasing the `trimming-interval` to reduce the amount of time the primary server spends broadcasting trimming messages and to reduce the amount of time the secondary servers spend processing these messages.

Decreasing the `trimming-interval` may increase recovery time as the new primary server's in-memory queue will be more out of date relative to the old primary.

For more information, see [Table 24–11, "Child Elements of ha-broadcast-adapter for the Broadcast Output Adapter"](#).

## Oracle Coherence Performance Tuning Options

When configuring Oracle Coherence in a high-availability architecture, consider the following performance tuning options:

- [Section, "Oracle Coherence Heartbeat Frequency"](#)
- [Section, "Oracle Coherence Serialization"](#)

For more information, see "Performance Tuning" in the *Oracle Coherence Developer's Guide* at [http://download.oracle.com/docs/cd/E15357\\_01/coh.360/e15723/tune\\_perftune.htm](http://download.oracle.com/docs/cd/E15357_01/coh.360/e15723/tune_perftune.htm).

### Oracle Coherence Heartbeat Frequency

To reduce failover time, increase Coherence heartbeat timeout machine frequency and reduce the number of heartbeats before failure.

### Oracle Coherence Serialization

To improve messaging performance, implement the Oracle Coherence Portable Object Format (POF) for serialization. POF is a language agnostic binary format that was designed to be very efficient in both space and time. Using POF instead of Java serialization can greatly improve performance.

For more information, see <http://coherence.oracle.com/display/COH35UG/The+Portable+Object+Format>.



# Part V

---

## Appendices

Part V contains the following appendices:

- [Appendix A, "Additional Information about Spring and OSGi"](#)
- [Appendix B, "Oracle Event Processing Schemas"](#)
- [Appendix C, "Schema Reference: EPN Assembly spring-wlevs-v11\\_1\\_1\\_6.xsd"](#)
- [Appendix D, "Schema Reference: Component Configuration wlevs\\_application\\_config.xsd"](#)
- [Appendix E, "Schema Reference: Deployment deployment.xsd"](#)
- [Appendix F, "Schema Reference: Server Configuration wlevs\\_server\\_config.xsd"](#)
- [Appendix G, "Schema Reference: Message Catalog msgcat.dtd"](#)
- [Appendix H, "Schema Reference: Locale Message Catalog l10n\\_msgcat.dtd"](#)
- [Appendix I, "Oracle Event Processing Metadata Annotation Reference"](#)





---

---

## Additional Information about Spring and OSGi

This appendix lists links to more information about the Spring Framework and OSGi Service Platform, on which Oracle Event Processing applications are built.

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](http://www.osgi.org/osgi_technology/download_specs.asp?section=2#Release4)



---

---

## Oracle Event Processing Schemas

This appendix introduces schemas behind Oracle Event Processing configuration and deployment XML files, including brief examples of each.

This appendix includes the following sections:

- [EPN Assembly Schema `spring-wlevs-v11\_1\_1\_6.xsd`](#)
- [Component Configuration Schema `wlevs\_application\_config.xsd`](#)
- [Deployment Schema `deployment.xsd`](#)
- [Server Configuration Schema `wlevs\_server\_config.xsd`](#)

### EPN Assembly Schema `spring-wlevs-v11_1_1_6.xsd`

You use the EPN assembly file to declare the components that make up your Oracle Event Processing 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 Event Processing-specific services.

The `spring-wlevs-v11_1_1_6.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 Event Processing installation directory, such as `d:\oracle_cep`.

For more information, see [Appendix C, "Schema Reference: EPN Assembly `spring-wlevs-v11\_1\_1\_6.xsd`"](#).

### 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_1_1_6.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 Event Processing 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>

```

## Component Configuration Schema wlevs\_application\_config.xsd

An Oracle Event Processing 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 Event Processing 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 Event Processing installation directory, such as `d:\oracle_cep`.

For more information, see [Appendix D, "Schema Reference: Component Configuration wlevs\\_application\\_config.xsd"](#).

### 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>

```

## Deployment Schema deployment.xsd

The deployment file for an Oracle Event Processing 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 Event Processing installation directory, such as `d:\oracle_cep`.

For more information, see [Appendix E, "Schema Reference: Deployment deployment.xsd"](#).

## 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>

```

## Server Configuration Schema wlevs\_server\_config.xsd

The Oracle Event Processing 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 Event Processing instance, you can update this file manually and add or remove server configuration elements.

The `wlevs_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 Event Processing installation directory, such as `d:\oracle_cep`.

For more information, see [Appendix F, "Schema Reference: Server Configuration wlevs\\_server\\_config.xsd"](#).

## 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 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>
```





---

---

## Schema Reference: EPN Assembly spring-wlevs-v11\_1\_1\_6.xsd

This appendix provides a reference to elements of the `spring-wlevs-v11_1_1_6.xsd` schema, the schema behind assembly XML files with which you declare the components that make up your Oracle Event Processing event processing networks (EPNs).

This appendix includes the following sections:

- [Overview of the Oracle Event Processing Application Assembly Elements](#)
- `wlevs:adapter`
- `wlevs:application-timestamped`
- `wlevs:cache`
- `wlevs:cache-listener`
- `wlevs:cache-loader`
- `wlevs:cache-source`
- `wlevs:cache-store`
- `wlevs:caching-system`
- `wlevs:channel`
- `wlevs:class`
- `wlevs:event-bean`
- `wlevs:event-type`
- `wlevs:event-type-repository`
- `wlevs:expression`
- `wlevs:factory`
- `wlevs:function`
- `wlevs:instance-property`
- `wlevs:listener`
- `wlevs:metadata`
- `wlevs:processor`
- `wlevs:properties`
- `wlevs:property`

- [wlevs:property](#)
- [wlevs:source](#)
- [wlevs:table](#)
- [wlevs:table-source](#)

## Overview of the Oracle Event Processing Application Assembly Elements

Oracle Event Processing 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.

### Element Hierarchy

The Oracle Event Processing 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:class
      wlevs:metadata
      wlevs:properties
      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](#)

## Example of an EPN Assembly File That Uses Oracle Event Processing Elements

The following sample EPN assembly file from the HelloWorld application shows how to use many of the Oracle Event Processing 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: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_1_1_6.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>
```

## wlevs:adapter

Use this element to declare an adapter component to the Spring application context.

### Child Elements

The `wlevs:adapter` application assembly element supports the following child elements:

- [wlevs:listener](#)
- [wlevs:instance-property](#)
- [wlevs:property](#)

### Attributes

[Table C-1](#) lists the attributes of the `wlevs:adapter` application assembly element.

**Table C-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 <code>true</code> and <code>false</code> . Default value is <code>false</code> .	Boolean	No.
listeners	Specifies the components that listen to this component. Set this attribute to the value of the <code>id</code> attribute of the element that declared the component.	String	No.
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 <code>csvgen</code> or <code>loadgen</code> utilities to simulate a data feed, use the hard-coded <code>csvgen</code> or <code>loadgen</code> values, respectively, such as:  <code>provider="csvgen"</code>  If you are using one of the built-in HTTP publish-subscribe adapters, then specify the following hard-coded values: <ul style="list-style-type: none"><li>■ For the built-in pub-sub adapter used for <i>publishing</i>, specify the hard-coded <code>httppub</code> value, such as: <code>provider="httppub"</code></li><li>■ For the built-in pub-sub adapter used for <i>subscribing</i>, specify the hard-coded <code>httpsub</code> value, such as: <code>provider="httpsub"</code></li></ul> If you are using a JMS adapter, then specify one of the following hard-coded values: <ul style="list-style-type: none"><li>■ For the inbound JMS adapter, specify the <code>jms-inbound</code> value, such as: <code>provider="jms-inbound"</code></li><li>■ For the outbound JMS adapter, specify the <code>jms-outbound</code> value, such as: <code>provider="jms-outbound"</code></li></ul> You must specify either the <code>provider</code> or <code>class</code> 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 <code>provider</code> or <code>class</code> 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 <code>onEvent</code> method.  Oracle Event Processing 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 <code>init</code> method.  Oracle Event Processing 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

**Table C-1 (Cont.) Attributes of the wlevs:adapter Application Assembly Element**

Attribute	Description	Data Type	Required?
activate-method	Specifies the method of the adapter implementation that corresponds to the lifecycle <code>activate</code> method.  Oracle Event Processing 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 <code>suspend</code> method.  Oracle Event Processing invokes this method when the application is suspended.	String	No
destroy-method	Specifies the method of the adapter implementation that corresponds to the lifecycle <code>destroy</code> method.  Oracle Event Processing invokes this method when the application is stopped.	String	No

## 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.

## 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 Event Processing server is responsible for assigning a timestamp to each event using `System.nanoTime`.

## 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.

## Attributes

Table C-2 lists the attributes of the `wlevs:application-timestamped` application assembly element.

**Table C–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 <i>Oracle Fusion Middleware CQL Language Reference for Oracle Event Processing</i> .	Boolean	No.

## 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.

## wlevs:cache

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

### 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.

## Attributes

Table C-3 lists the attributes of the `wlevs:cache` application assembly element.

**Table C-3 Attributes of the `wlevs:cache` Application Assembly Element**

Attribute	Description	Data Type	Required?
<code>id</code>	Unique identifier for this component. This identifier must correspond to the <code>&lt;name&gt;</code> element in the XML configuration file for this cache.	String	Yes.
<code>name</code>	Specifies an alternate name for this cache. If not specified, then the name of the cache is the same as its <code>id</code> attribute.	String	No.
<code>key-properties</code>	Specifies a comma-separated list of names of the properties that together form the unique key value for the objects in the cache, or <i>cache key</i> . 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 Event Processing inserts events that reach the cache using the unique key value as a key.  If you specify a key class using the <code>key-class</code> attribute, then this attribute is optional. If you specify neither <code>key-properties</code> nor <code>key-class</code> , then Oracle Event Processing uses the event object itself as both the key and value when it inserts the event object into the cache.	String	No.
<code>key-class</code>	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 <code>key-properties</code> attribute, then all properties on the <code>key-class</code> are assumed to be key properties. If you specify neither <code>key-properties</code> nor <code>key-class</code> , then Oracle Event Processing uses the event object itself as both the key and value when it inserts the event object into the cache.	String	No.
<code>value-type</code>	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.
<code>caching-system</code>	Specifies the caching system in which this cache is contained.  The value of this attribute corresponds to the <code>id</code> attribute of the appropriate <code>wlevs:caching-system</code> element.	String	Yes.
<code>advertise</code>	Advertises this service in the OSGi registry.  Valid values are <code>true</code> and <code>false</code> . Default value is <code>false</code> .	Boolean	No.

## 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`.

## 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`.

### Attributes

Table C-4 lists the attributes of the `wlevs:cache-listener` application assembly element.

**Table C-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 <code>id</code> attribute of the listening component. The listening component can be an adapter or a Spring bean.	String	No.

### Example

The following example shows how to use the `wlevs:cache-listener` 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.cache.jcache.CacheListener` interface. You must program the `wlevs.example.MyCacheListener` class yourself.

## wlevs:cache-loader

spring-wlevs-v11\_1\_1\_6.xsd Specifies the Spring bean that implements an object that loads data into a cache.

This element is always a child of `wlevs:cache`.

### Attributes

Table C-5 lists the attributes of the `wlevs:cache-loader` application assembly element.

**Table C-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 <code>id</code> attribute of the Spring bean. The Spring bean must implement the <code>com.bea.cache.jcache.CacheLoader</code> interface.	String	Yes.



## 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.

## 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.

## Attributes

Table C-6 lists the attributes of the `wlevs:cache-source` application assembly element.

**Table C-6** Attributes of the `wlevs:cache-source` Application Assembly Element

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

## 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:cache-source>
</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.

## 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`.

### Attributes

Table C-7 lists the attributes of the `wlevs:cache-store` application assembly element.

**Table C-7** Attributes of the `wlevs:cache-store` Application Assembly Element

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

### 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.

## wlevs:caching-system

Specifies the caching system used by the application.

### Child Elements

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

- `wlevs:instance-property`
- `wlevs:property`

### Attributes

Table C-8 lists the attributes of the `wlevs:caching-system` application assembly element.

**Table C-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:  <pre>&lt;wlevs:caching-system id="myCachingSystem" provider=coherence" /&gt;</pre> 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 the Oracle Event Processing native 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 the Oracle Event Processing native caching implementation.  If you specify this attribute, it is assumed that the third-party implementation code resides inside the Oracle Event Processing 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 the Oracle Event Processing native caching implementation for local single-JVM caches; this implementation uses an in-memory store.	String	No

## Example

The following example shows the simplest use of the wlevs: caching-system 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.

## 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](#).

## 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`

## Attributes

Table C-9 lists the attributes of the `wlevs:channel` application assembly element.

**Table C-9 Attributes of the `wlevs:channel` Application Assembly Element**

Attribute	Description	Data Type	Required?
<code>advertise</code>	Advertises this service in the OSGi registry. Valid values are <code>true</code> and <code>false</code> . Default value is <code>false</code> .	Boolean	No.
<code>batching</code>	Specifies whether batching of events should be enabled for the event channel. Valid values are <code>true</code> and <code>false</code> . Default value is <code>false</code> . For more information, see <a href="#">Section , "Batch Processing Channels"</a> .	Boolean	No.
<code>event-type</code>	Specifies the type of events that are allowed to pass through the event channel.	String	Yes.
<code>id</code>	Unique identifier for this component. This identifier must correspond to the <code>&lt;name&gt;</code> element in the XML configuration file for this channel, if one exists.	String	Yes.
<code>is-relation</code>	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 <code>false</code> .	Boolean	No.
<code>listeners</code>	Specifies the components that listen to this component. Separate multiple components using commas. Set this attribute to the value of the <code>id</code> attribute of the element ( <code>wlevs:adapter</code> , <code>wlevs:channel</code> , or <code>wlevs:processor</code> ) that defines the listening component.	String	No.
<code>max-size</code>	Specifies the maximum size of the FIFO buffer for this channel as <code>max-size</code> number of events. When <code>max-size = 0</code> , the channel synchronously passes-through events. When <code>max-size &gt; 0</code> , the channel processes events asynchronously, buffering events by the requested size. If <code>max-threads</code> is zero, then <code>max-size</code> is zero. The default value for this attribute is 1024.	integer	No.

**Table C-9 (Cont.) Attributes of the wlevs:channel Application Assembly Element**

Attribute	Description	Data Type	Required?
max-threads	<p>Specifies the maximum number of threads that will be used to process events for this channel.</p> <p>When <code>max-threads = 0</code>, the channel acts as a pass-through. Event ordering is preserved.</p> <p>When <code>max-threads &gt; 0</code>, 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 <code>max-size</code>. There will be up to <code>max-threads</code> number of threads consuming events from the queue. Event ordering is non-deterministic.</p> <p>You can change <code>max-threads</code> from 0 to a positive integer (that is, from a pass through to multiple threads) without redeploying. However, if you change <code>max-threads</code> from a positive integer to 0 (that is, from multiple threads to a pass through), then you must redeploy your application.</p> <p>If the <code>max-size</code> attribute is 0, then setting a value for <code>max-threads</code> has no effect.</p> <p>The default value for this attribute is 1.</p>	integer	No.
primary-key	<p>Specifies the primary key of a relation, as a list of event property names, separated by ", " or white-spaces.</p> <p>For more information, see <a href="#">Section , "Channels as Relations"</a>.</p>	String	No.
provider	<p>Specifies the streaming provider.</p> <p>Valid values are:</p> <ul style="list-style-type: none"> <li>▪ <code>oracle.channel</code></li> </ul> <p>Default value is <code>oracle.channel</code>, which is the out-of-the-box streaming provider.</p>	String	No.
source	<p>Specifies the component from which the channel sources events.</p> <p>Set this attribute to the value of the <code>id</code> attribute of the element (<code>wlevs:adapter</code>, <code>wlevs:channel</code>, or <code>wlevs:processor</code>) that defines the source component.</p>	String	No.

## 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`.

## wlevs:class

Use this element to specify the fully-qualified name of the JavaBean class to use as an event type implementation. This element must be a child of the `wlevs:event-type` element.

## Example

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

```
<wlevs:event-type-repository>
  <wlevs:event-type type-name="SimpleEvent">
    <wlevs:class>com.example.myapp.MyEventType</wlevs:class>
  </wlevs:event-type>
</wlevs:event-type-repository>
```

```

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

```

## 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 Event Processing 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 Event Processing.

For example, you can monitor an event bean using the Oracle Event Processing 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 Event Processing bean lifecycle by specifying methods in its EPN assembly file declaration, rather than by implementing Oracle Event Processing API interfaces.

## Child Elements

The `wlevs:event-bean` application assembly element supports the following child elements:

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

## Attributes

Table C–10 lists the attributes of the `wlevs:event-bean` application assembly element.

**Table C–10** Attributes of the `wlevs:event-bean` Application Assembly Element

Attribute	Description	Data Type	Required?
<code>id</code>	Unique identifier for this component. This identifier must correspond to the <code>&lt;name&gt;</code> element in the XML configuration file for this event-bean, if one exists.	String	Yes.
<code>advertise</code>	Advertises this service in the OSGi registry. Valid values are <code>true</code> and <code>false</code> . Default value is <code>false</code> .	Boolean	No.
<code>listeners</code>	Specifies the components that listen to this component. Set this attribute to the value of the <code>id</code> attribute of the element that declared the component.	String	No.
<code>class</code>	Specifies the Java class that implements this event bean. The bean is not required to implement any Oracle Event Processing interfaces. You must specify either the <code>provider</code> or <code>class</code> attribute, but not both, otherwise an exception is raised.		
<code>provider</code>	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 <code>provider</code> or <code>class</code> attribute, but not both, otherwise an exception is raised.	String	No.

**Table C-10 (Cont.) Attributes of the wlevs:event-bean Application Assembly Element**

Attribute	Description	Data Type	Required?
onevent-method	Specifies the method of the event bean implementation that corresponds to the lifecycle <code>onEvent</code> method. Oracle Event Processing 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 Event Processing interface.	String	No
init-method	Specifies the method of the event bean implementation that corresponds to the lifecycle <code>init</code> method. Oracle Event Processing 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 Event Processing interface.	String	No
activate-method	Specifies the method of the event bean implementation that corresponds to the lifecycle <code>activate</code> method. Oracle Event Processing 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 Event Processing interface.	String	No
suspend-method	Specifies the method of the event bean implementation that corresponds to the lifecycle <code>suspend</code> method. Oracle Event Processing 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 Event Processing interface.	String	No
destroy-method	Specifies the method of the event bean implementation that corresponds to the lifecycle <code>destroy</code> method. Oracle Event Processing 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 Event Processing interface.	String	No

## 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.

## wlevs:event-type

Specifies the definition of an event type used in the Oracle Event Processing 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 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.
- Specify event type properties declaratively by using a `wlevs:properties` child element.

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

The best practice is to define your event type by using the `wlevs:class` child element because you can then reuse the specified JavaBean class, and you control exactly what the event type looks like.

### Child Elements

The `wlevs:event-type` application assembly element supports the following child elements:

- `wlevs:class`
- `wlevs:metadata` (deprecated)
- `wlevs:properties`
- `wlevs:property`

### Attributes

Table C–11 lists the attributes of the `wlevs:event-type` application assembly element.

**Table C–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 Event Processing 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.
object-support	Specifies if Java objects should be fully supported. Allowable values are <code>true</code> , <code>false</code> , and <code>object-relational</code> ; default is <code>object-relational</code> . If set to <code>false</code> , then the Java primitive wrappers (for example, <code>java.lang.Integer</code> ) and <code>java.lang.String</code> are treated solely as primitive types. If set to <code>true</code> , then Java primitive wrappers are treated as classes. If set to <code>object-relational</code> ,	String	No.

### 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 , "Choosing a Data Type for an Event Type"](#).

## 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.

### Child Elements

The `wlevs:event-type-repository` application assembly element supports the following child element:

- [wlevs:event-type](#)

### 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 , "wlevs:event-type"](#) for additional details.

## 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 Fusion Middleware CQL Language Reference for Oracle Event Processing*.

## 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.

## 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.

## Attributes

Table C–12 lists the attributes of the `wlevs:factory` application assembly element.

**Table C–12** Attributes of the `wlevs:factory` Application Assembly Element

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

## Example

The following example shows how to use the `wlevs:factory` 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`.

## wlevs:function

Use this element to specify a bean that contains user-defined functions for a processor. Oracle Event Processing 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:

- "User-Defined Functions" in the *Oracle Fusion Middleware CQL Language Reference for Oracle Event Processing*
- "EPL Reference: Functions" in the *Oracle Fusion Middleware EPL Language Reference for Oracle Event Processing*

## Attributes

[Table C-13](#) lists the attributes of the `wlevs:function` application assembly element.

**Table C-13 Attributes of the `wlevs:function` Application Assembly Element**

Attribute	Description	Data Type	Required?
<code>exec-method</code>	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. In this case, the method must be public and must be uniquely identifiable by its name (that is, the method cannot have been overridden).  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.
<code>function-name</code>	For a user-defined single-row function on an Oracle CQL processor, use this attribute to define an alias for the <code>exec-method</code> name. You can then use the <code>function-name</code> in your Oracle CQL query instead of the <code>exec-name</code> .  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 <code>function-name</code> 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 <code>function-name</code> in your EPL query instead of the Spring bean class name.  The default value is the Spring bean name.	String	No.

**Table C-13 (Cont.) Attributes of the wlevs:function Application Assembly Element**

Attribute	Description	Data Type	Required?
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.

## Example

The following examples show how to use the wlevs:function element and its attributes on both Oracle CQL and EPL processors:

- [Section , "Single-Row User-Defined Function on an Oracle CQL Processor"](#)
- [Section , "Single-Row User-Defined Function on an EPL Processor"](#)
- [Section , "Aggregate User-Defined Function on an Oracle CQL Processor"](#)
- [Section , "Aggregate User-Defined Function on an EPL Processor"](#)
- [Section , "Specifying the Implementation Class: Nested Bean or Reference"](#)

### Single-Row User-Defined Function on an Oracle CQL Processor

[Example C-1](#) shows how you implement a single-row user-defined the function for an Oracle CQL processor.

#### **Example C-1 Single-Row User Defined Function Implementation Class**

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

public class MyMod {
    public Object execute(int arg0, int arg1) {
        return new Integer(arg0 % arg1);
    }
}
```

[Example C-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 C-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 C-3](#) shows how you invoke the function in an Oracle CQL query.

#### **Example C-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>
...

```

## Single-Row User-Defined Function on an EPL Processor

[Example C-4](#) shows how you implement a single-row user-defined the function for an EPL processor.

### **Example C-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 C-5](#) shows how to use the `wlevs:function` to define a single-row function for an EPL processor in the EPN assembly file.

### **Example C-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 C-6](#) shows how you invoke the function in an EPL query.

### **Example C-6 Invoking the Single-Row User-Defined Function on an EPL Processor**

```

<rule><![CDATA[
    select math.square(inputValue) ...
]]></rule>

```

## Aggregate User-Defined Function on an Oracle CQL Processor

[Example C-7](#) shows how to implement a user-defined aggregate function for an Oracle CQL processor.

### **Example C-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;
}
}
```

[Example C-8](#) shows how to use the `wlevs:function` to define an aggregate function on an Oracle CQL processor in the EPN assembly file.

**Example C–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>

```

[Example C–9](#) shows how you invoke the function in an Oracle CQL query.

**Example C–9 Invoking the Aggregate User-Defined Function on an Oracle CQL Processor**

```

...
<query id="uda6"><![CDATA[
  select var(c2) from S4[range 3]
]]></query>
...

```

**Aggregate User-Defined Function on an EPL Processor**

[Example C–10](#) shows how to implement a user-defined aggregate function for an EPL processor.

**Example C–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 C-11](#) shows how to use the `wlevs:function` to define an aggregate function on an EPL processor in the EPN assembly file.

#### **Example C-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 C-12](#) shows how you invoke the function in an EPL query.

#### **Example C-12 Invoking the Aggregate User-Defined Function on an EPL Processor**

```

...
<rule><![CDATA[
  select var(c2) from S4
]]></rule>
...

```

### **Specifying the Implementation Class: Nested Bean or Reference**

[Example C-13](#) shows how to use the `wlevs:function` element with a nested bean element in the EPN assembly file.



**Example C-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 C-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 C-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"/>

```

## 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.

For example, when you specify an `wlevs:instance-property` for a `wlevs:event-bean`, Oracle Event Processing will look for a corresponding setter method on the Java class you implement, not on the `com.bea.wlevs.spring.EventBeanFactoryBean` that instantiates your class. To specify a property on the factory, use `wlevs:property`

This element is used only as a child of `wlevs:adapter`, `wlevs:event-bean`, `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>.

## 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

## Attributes

Table C–14 lists the attributes of the `wlevs:instance-property` application assembly element.

**Table C–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 <code>&lt;ref bean='...' /&gt;</code> element.	String	No.
value	A short-cut alternative to a nested <code>&lt;value&gt;...&lt;/value&gt;</code> element.	String	No.

## Example

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

```
<wlevs:event-bean id="pubsubCounterBeanRemote"
  class="com.oracle.cep.example.httppubsub.RemoteEventCounter">
  <wlevs:listener ref="pubsubRemote" />
  <wlevs:instance-property name="expectedEvents" value="4000" />
</wlevs:event-bean>
```

In the example, the event bean `com.oracle.cep.example.httppubsub.RemoteEventCounter` class provides an appropriate setter method:

```
...
private int expectedEvents;

public void setExpectedEvents(String expectedEvents) {
    this.expectedEvents = new Integer(expectedEvents).intValue();
}
...
```

Note that the `instance-property` is of type `String`. Your setter method must convert this if necessary. In this example, the `String` is converted to an `int` value.

The name of the setter method must conform to JavaBean naming conventions. In this example, the setter name is `setExpectedEvents` and this corresponds to the `wlevs:instance-property` element name attribute value `expectedEvents`, according to JavaBean conventions. If the name attribute value is `obj` and the setter method name is `setObject`, Oracle Event Processing will throw an `Invalid Property` exception. In this case, the setter name should be `setObj`.

## 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`.

## Attributes

Table C–15 lists the attributes of the `wlevs:listener` application assembly element.

**Table C–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 <code>id</code> attribute of the listener component. You do not specify this attribute if you are nesting listeners.	String	No.

## Example

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

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

In the example, the `helloworldOutstream` 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`.

## 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 Event Processing 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>.

## 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>.

## Attributes

Table C–16 lists the attributes of the `wlevs:metadata` application assembly element.

**Table C–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 <i>only</i> if you have nested entry elements.	String	No.

## 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.

## wlevs:processor

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

### 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`

### Attributes

[Table C–17](#) lists the attributes of the `wlevs:processor` application assembly element.

**Table C–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 <code>&lt;name&gt;</code> element in the XML configuration file for this processor; this is how Oracle Event Processing knows which Oracle CQL or EPL rules to execute for which processor component in your network.	String	Yes.

**Table C–17 (Cont.) Attributes of the wlevs:processor Application Assembly Element**

Attribute	Description	Data Type	Required?
advertise	Advertises this service in the OSGi registry. Valid values are <code>true</code> and <code>false</code> . Default value is <code>false</code> .	Boolean	No.
listeners	Specifies the components that listen to this component. Set this attribute to the value of the <code>id</code> 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: <ul style="list-style-type: none"> <li>▪ <code>epl</code></li> <li>▪ <code>cql</code></li> </ul> The default value is <code>cql</code> .	String	No.
queryURL	Specifies a URL that points to an Oracle CQL or EPL rules definition file for this processor.	String.	No.

## 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 Event Processing knows which Oracle CQL or EPL rules it must file for this particular processor.

## wlevs:properties

Defines the list of properties for an event type. Use this element when you are defining an event type declaratively, such as for a type based on a tuple or `java.util.Map`. For an event type created from a JavaBean class, allow properties to be defined by accessor methods in the class.

## Child Elements

The `wlevs:properties` application assembly element supports the following child elements:

- [wlevs:property](#)

## Attributes

[Table C–18](#) lists the attributes of the `wlevs:event-type` application assembly element.

**Table C–18 Attributes of the wlevs:properties Application Assembly Element**

Attribute	Description	Data Type	Required?
type	Specifies the type that this event type should be created from. Allowable values are <code>tuple</code> and <code>map</code> ; default is <code>tuple</code> . If this attribute's value is <code>map</code> , then Oracle Event Processing will instantiate the event type as a <code>java.util.Map</code> instance. Otherwise, the type will be instantiated as a tuple.	String	No.

## Example

The following example shows how to use the `wlevs:properties` 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 , "Choosing a Data Type for an Event Type"](#).

## wlevs:property

Defines the property of an event type that you create declaratively, such as an event type based on a tuple or `java.util.Map`. You use this `wlevs:property` element as a child of the `wlevs:properties` element.

Note that this element is different from the `wlevs:property` element that is an extension of the Spring beans property element. This element must always be used as a child of the `wlevs:properties` element.

## Attributes

[Table C–19](#) lists the attributes of the `wlevs:property` application assembly element.

**Table C–19 Attributes of the `wlevs:property` Application Assembly Element**

Attribute	Description	Data Type	Required?
name	Name of the event property.	String	Yes.
type	Type of the event property. If this event type is defined as a tuple, then the type attribute value must be one of the OCEP native types defined by <code>com.bea.wlevs.ede.api.Type</code> . Those include <code>bigint</code> , <code>boolean</code> , <code>byte</code> , <code>char</code> , <code>double</code> , <code>float</code> , <code>int</code> , <code>interval</code> , <code>object</code> , <code>sqlxml</code> , <code>timestamp</code> , <code>unknown</code> , <code>xmltype</code> . If this event type is defined as a map, then the type attribute value is the fully qualified name of a Java class that must be available in the class loader of the application that defines the event type. The string used as the Java class name must conform to the same rules as <code>Class.forName()</code> . In addition, Java primitives can be used (e.g. <code>int</code> , <code>float</code> ). Finally, you can specify an array by appending the characters <code>[]</code> to the Java class name	String	Yes.
length	If the property is of array type, this specifies the length of the array. If the type is not an array and length is specified, it will be ignored.	String	No.

## Example

The following example shows how to use the `wlevs:property` 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.

## wlevs:property

Specifies a custom property to apply to the event type.

For example, when you specify a `wlevs:property` for a `wlevs:event-bean`, Oracle Event Processing will look for a corresponding setter method on the `com.bea.wlevs.spring.EventBeanFactoryBean` that instantiates your Java class, not on the Java class you implement. To specify a property on your Java class, use [wlevs:instance-property](#).

This element is used only as a child of [wlevs:adapter](#), [wlevs:event-bean](#), [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>.

## 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`

## Attributes

Table C–20 lists the attributes of the `wlevs:property` application assembly element.

**Table C–20 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 <code>&lt;ref bean='...' /&gt;</code> element.	String	No.
value	A short-cut alternative to a nested <code>&lt;value&gt;...&lt;/value&gt;</code> element.	String	No.

## 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`.

## 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`.

## Attributes

Table C–21 lists the attributes of the `wlevs:source` application assembly element.



**Table C–21 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.

## 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.

## 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.

## Attributes

[Table C–22](#) lists the attributes of the `wlevs:table` application assembly element.

**Table C–22 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 <code>&lt;name&gt;</code> 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 Event Processing server configuration file used to access this database table.	String	Yes.

## 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.

## 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.

### Attributes

Table C–23 lists the attributes of the `wlevs:table-source` application assembly element.

**Table C–23** *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 <code>id</code> attribute of a <code>wlevs:table</code> element.	String	Yes.

### 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>
```

---

---

## Schema Reference: Component Configuration wlevs\_application\_config.xsd

This appendix provides a reference to the elements of the `wlevs_application_config.xsd` schema, the schema behind XML files you use to configure Oracle Event Processing application components such as adapters, channels, caching systems, and event beans.

This appendix includes the following sections:

- [Overview of the Oracle Event Processing Component Configuration Elements](#)
- [accept-backlog](#)
- [active](#)
- [adapter](#)
- [amount](#)
- [application](#)
- [average-interval](#)
- [average-latency](#)
- [batch-size](#)
- [batch-time-out](#)
- [binding](#)
- [bindings \(jms-adapter\)](#)
- [bindings \(processor\)](#)
- [buffer-size](#)
- [buffer-write-attempts](#)
- [buffer-write-timeout](#)
- [cache](#)
- [caching-system](#)
- [channel](#)
- [channel \(http-pub-sub-adapter Child Element\)](#)
- [channel-name](#)
- [coherence-cache-config](#)
- [coherence-caching-system](#)

- 
- coherence-cluster-config
  - collect-interval
  - concurrent-consumers
  - connection-jndi-name
  - connection-encrypted-password
  - connection-password
  - connection-user
  - database
  - dataset-name
  - delivery-mode
  - destination-jndi-name
  - destination-name
  - destination-type
  - diagnostic-profiles
  - direction
  - durable-subscription
  - durable-subscription-name
  - duration
  - enabled
  - encrypted-password
  - end
  - end-location
  - event-bean
  - event-type
  - event-type-list
  - eviction-policy
  - fail-when-rejected
  - group-binding
  - heartbeat
  - http-pub-sub-adapter
  - idle-time
  - inject-parameters
  - jms-adapter
  - jndi-factory
  - jndi-provider-url
  - listeners
  - location

- 
- max-latency
  - max-size
  - max-threads
  - message-selector
  - name
  - netio
  - num-threads
  - offer-timeout
  - param
  - parameter
  - params
  - partition-order-capacity
  - password
  - playback-parameters
  - playback-speed
  - processor (EPL)
  - processor (Oracle CQL)
  - profile
  - provider-name
  - query
  - record-parameters
  - repeat
  - rule
  - rules
  - schedule-time-range
  - schedule-time-range-offset
  - selector
  - server-context-path
  - server-url
  - session-ack-mode-name
  - session-transacted
  - stage
  - start
  - start-location
  - start-stage
  - store-policy-parameters
  - stream

- [symbol](#)
- [symbols](#)
- [threshold](#)
- [throughput](#)
- [throughput-interval](#)
- [time-range](#)
- [time-range-offset](#)
- [time-to-live](#)
- [trace-parameters](#)
- [unit](#)
- [user](#)
- [value](#)
- [view](#)
- [work-manager](#)
- [work-manager-name](#)
- [write-behind](#)
- [write-none](#)
- [write-through](#)

## Overview of the Oracle Event Processing Component Configuration Elements

Oracle Event Processing 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.

### Element Hierarchy

The top-level Oracle Event Processing component configuration elements are organized into the following hierarchy:

- [config](#)
  - [adapter](#) (see [Example D-1](#))
  - [http-pub-sub-adapter](#) (see [Example D-2](#))
  - [jms-adapter](#) (see [Example D-3](#))
  - [processor](#) (see [Example D-4](#) and [Example D-5](#))
  - [stream](#) (see [Example D-6](#))
  - [channel](#) (see [Example D-7](#))
  - [event-bean](#) (see [Example D-8](#))
  - [caching-system](#) (see [Example D-9](#))
  - [coherence-caching-system](#) (see [Example D-10](#))
  - [diagnostic-profiles](#) (see [Example D-11](#))

**Example D-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 D-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 D-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
connection-user
One of:
  connection-password
  connection-encrypted-password
work-manager
concurrent-consumers

```

```

message-selector
session-ack-mode-name
session-transacted
delivery-mode
bindings (jms-adapter)
    group-binding
        param
destination-type
durable-subscription
durable-subscription-name
    
```

**Example D-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 (processor)
    binding
      params

```

**Example D–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
  bindings (processor)
    binding

```

params

**Example D-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 D-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
  inject-parameters
  trace-parameters
  max-size
  max-threads
  selector
  heartbeat
  partition-order-capacity
  offer-timeout
  fail-when-rejected

```

**Example D–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 D-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 D-10 coherence-caching-system Element Hierarchy**

```

coherence-caching-system
  name
  coherence-cache-config
  coherence-cluster-config

```

**Example D-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
    threshold
  throughput
    name
    throughput-interval
      amount
      unit
    average-interval
      amount
      unit
    location
      application
      stage
      direction

```

**Example of an Oracle Event Processing Component Configuration File**

The following sample component configuration file from the HelloWorld application shows how to use many of the Oracle Event Processing elements:

```
<?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>
```

## 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.

### Child Elements

The `accept-backlog` component configuration element has no child elements.

### Attributes

The `accept-backlog` component has no attributes.

### 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>
```

## active

Specify `true` for this element to specify that event tracing or event injection is on. The default is `true`.

Note that when the value of the `active` element is `false`, the `channel-name` value will be ignored.

For more on event tracing and injection, see the following:

- [Section , "Tracing Events"](#)
- [Section , "Injecting Events"](#)



## Child Elements

The active component configuration element has no child elements.

## Attributes

The active component configuration element has no attributes.

## Example

The component configuration excerpt shown in the following example illustrates how you might configure a processor for event tracing. The `trace-parameters` element's `active` child element specifies that tracing is on, while the `channel-name` element specifies the HTTP pub-sub channel to which traced elements should be sent.

```
<processor>
  <name>FindCrossRates</name>
  <trace-parameters>
    <active>true</active>
    <channel-name>/NonClusteredServer/fx/FindCrossRates/output</channel-name>
  </trace-parameters>
  <rules>
    <!-- Query rules omitted. -->
  </rules>
</processor>
```

## 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 15, "Integrating an External Component Using a Custom Adapter"](#).

## Child Elements

The adapter component configuration element supports the following child elements:

- `name`
- `record-parameters`
- `playback-parameters`
- `symbols`
- `work-manager-name`
- `netio`

## Attributes

The adapter component configuration element has no attributes.

## 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.

## 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 Fusion Middleware Visualizer User's Guide for Oracle Event Processing*.

## Child Elements

The amount component configuration has no child elements:

## Attributes

The amount component has no attributes.

## 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>
```

## application

Use this element to define the type of application Oracle Event Processing 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 Fusion Middleware Visualizer User's Guide for Oracle Event Processing*.

## Child Elements

The application component configuration has no child elements:

## Attributes

The application component has no attributes.

## 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>
```

## 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 Fusion Middleware Visualizer User's Guide for Oracle Event Processing*.

## Child Elements

The `average-interval` component configuration element supports the following child elements:

- `amount`
- `unit`

## Attributes

The `average-interval` component has no attributes.

## 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>
```

## 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 Fusion Middleware Visualizer User's Guide for Oracle Event Processing*.

## Child Elements

The `average-latency` component configuration element supports the following child elements:

- `name`
- `collect-interval`
- `start-location`
- `end-location`
- `threshold`

## Attributes

The average-latency component has no attributes.

## 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>
      <threshold>
        <amount>100</amount>
        <unit>MILLISECONDS</unit>
      </threshold>
    </average-latency>
  </profile>
</diagnostic-profiles>
```

## 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.

## Child Elements

The batch-size component configuration element has no child elements.

## Attributes

The batch-size component has no attributes.

## 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>
</record-parameters>
```

```
</store-policy-parameters>
<batch-size>1</batch-size>
<batch-time-out>10</batch-time-out>
</record-parameters>
```

## 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.

### Child Elements

The `batch-time-out` component configuration element has no child elements.

### Attributes

The `batch-time-out` component has no attributes.

### 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>
```

## binding

Use this element to define values for a parameterized Oracle CQL or EPL rule in an EPL processor component.

For more information, see:

- "Parameterized Queries" in the *Oracle Fusion Middleware CQL Language Reference for Oracle Event Processing*
- "Parameterized Queries" in the *Oracle Fusion Middleware EPL Language Reference for Oracle Event Processing*

### Child Elements

The `binding` component configuration element supports the following child elements:

- [params](#)

### Attributes

[Table D-1](#) lists the attributes of the `binding` component configuration element.

**Table D-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.

## 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>
```

## bindings (jms-adapter)

Using the `com.oracle.cep.cluster.hagroups.ActiveActiveGroupBean`, you can partition an incoming JMS stream in Oracle Event Processing applications by utilizing the notification groups that the `ActiveActiveGroupBean` creates.

Use this element to associate a notification group with a particular message-selector value.

For more information, see [Section , "ActiveActiveGroupBean"](#).

### Child Elements

The bindings component configuration element supports the following child elements:

- [group-binding](#)

### Attributes

The bindings component has no attributes.

## Example

The following example shows how to use the bindings element in the component configuration file:

```
<jms-adapter>
  <name>JMSInboundAdapter</name>
  <event-type>StockTick</event-type>
  <jndi-provider-url>t3://ppurich-pc:7001</jndi-provider-url>
  <destination-jndi-name>./Topic1</destination-jndi-name>
  <user>weblogic</user>
  <password>weblogic1</password>
  <work-manager>JettyWorkManager</work-manager>
  <concurrent-consumers>1</concurrent-consumers>
  <session-transacted>true</session-transacted>
  <message-selector>${CONDITION}</message-selector>
  <bindings>
    <group-binding group-id="ActiveActiveGroupBean_group1">
      <param id="CONDITION">acctid > 400</param>
    </group-binding>
    <group-binding group-id="ActiveActiveGroupBean_group2">
      <param id="CONDITION">acctid BETWEEN 301 AND 400</param>
    </group-binding>
    <group-binding group-id="ActiveActiveGroupBean_group3">
      <param id="CONDITION">acctid BETWEEN 201 AND 300</param>
    </group-binding>
    <group-binding group-id="ActiveActiveGroupBean_group4">
      <param id="CONDITION">acctid <= 200</param>
    </group-binding>
  </bindings>
</jms-adapter>
```

In this configuration, when the application is deployed to an Oracle Event Processing server with a cluster element groups child element that contains ActiveActiveGroupBean\_group1, then the CONDITION parameter is defined as acctid > 400 and the application processes events whose acctid property is greater than 400.

## bindings (processor)

Use this element to define bindings for one or more parameterized Oracle CQL or EPL rules in a processor component.

For more information, see:

- "Parameterized Queries" in the *Oracle Fusion Middleware CQL Language Reference for Oracle Event Processing*
- "Parameterized Queries" in the *Oracle Fusion Middleware EPL Language Reference for Oracle Event Processing*

## Child Elements

The bindings component configuration element supports the following child elements:

- [binding](#)

## Attributes

The bindings component has no attributes.



## 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>
```

## 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.

## Child Elements

The `buffer-size` component configuration element has no child elements.

## Attributes

The `buffer-size` component has no attributes.

## 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>
    </write-behind>
  </cache>
</caching-system>
```

```
        <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>
```

## 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.

### Child Elements

The `buffer-write-attempts` component configuration element has no child elements.

### Attributes

The `buffer-write-attempts` component has no attributes.

### 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>
```

## 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.

## Child Elements

The `buffer-write-timeout` component configuration element has no child elements.

## Attributes

The `buffer-write-timeout` component has no attributes.

## 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 >
```

## 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 Event Processing application; it is not necessary for the application to function correctly. Oracle Event Processing 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 , "Configuring an Oracle Event Processing Local Caching System and Cache"](#).

## 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](#)

## Attributes

The cache component has no attributes.

## 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>
```

## caching-system

Use this element to define an Oracle Event Processing 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 , "Configuring a Cache as an Event Listener"](#).

## Child Elements

The caching-system component configuration element supports the following child elements:

- [name](#)
- [cache](#)

## Attributes

The caching-system component has no attributes.

## Example

The following example shows how to use the `channel` element in the component configuration file:

```
<channel>
  <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>
</channel>
```

In the example, the channel's unique identifier is `providerCachingSystem`.

## channel

Use this element to define a channel component. An Oracle Event Processing 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).

### Child Elements

The `channel` component configuration element supports the following child elements:

- `name`
- `record-parameters`
- `playback-parameters`
- `max-size`
- `max-threads`
- `selector`
- `heartbeat`
- `partition-order-capacity`

### Attributes

The `channel` component has no attributes.

## 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`.

## 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.

### Child Elements

The `channel` component configuration element has no child elements.

### Attributes

The `channel` component has no attributes.

### 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`.

## channel-name

Use this element to specify the name of the channel onto which events should be injected or to which traced events should be sent. The element's value must be a path to the channel in a form like the following default values (note that the path must begin with a slash):

- For event tracing:  
*/serverID/appID/stageID/output*
- For event injection:  
*/serverID/appID/stageID/input*

Note that when the value of the `active` element is `false`, the `channel-name` value will be ignored.

For more on event tracing and injection, see the following:

- [Section , "Tracing Events"](#)
- [Section , "Injecting Events"](#)

### Child Elements

The `channel-name` component configuration element has no child elements.

## Attributes

The channel-name component configuration element has no attributes.

## Example

The component configuration excerpt shown in the following example illustrates how you might configure a processor for event tracing. The `trace-parameters` element's `active` child element specifies that tracing is on, while the `channel-name` element specifies the HTTP pub-sub channel to which traced elements should be sent.

```
<processor>
  <name>FindCrossRates</name>
  <trace-parameters>
    <active>true</active>
    <channel-name>/NonClusteredServer/fx/FindCrossRates/output</channel-name>
  </trace-parameters>
  <rules>
    <!-- Query rules omitted. -->
  </rules>
</processor>
```

## coherence-cache-config

Use this element to define the Oracle Coherence cache configuration for a [coherence-caching-system](#).

For more information, see [Section , "Configuring an Oracle Event Processing Local Caching System and Cache"](#).

## Child Elements

The coherence-cache-config component configuration element has no child elements.

## Attributes

The coherence-cache-config component has no attributes.

## 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>
```

## 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 , "Configuring an Oracle Coherence Caching System and Cache"](#).

## Child Elements

The `coherence-caching-system` component configuration element supports the following child elements:

- [name](#)
- [coherence-cache-config](#)
- [coherence-cluster-config](#)

## Attributes

The `coherence-caching-system` component has no attributes.

## 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`.

## 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 Event Processing Multi-Server Domain Administration" in the *Oracle Fusion Middleware Administrator's Guide for Oracle Event Processing*.

## Child Elements

The `coherence-cache-config` component configuration element has no child elements.

## Attributes

The `coherence-cache-config` component has no attributes.

## 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>
```



## 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 Fusion Middleware Visualizer User's Guide for Oracle Event Processing*.

### Child Elements

The `collect-interval` component configuration element supports the following child elements:

- `amount`
- `unit`

### Attributes

The `collect-interval` component has no attributes.

### 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>
```

## 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 because the converter bean will be shared among the consumers.

If `concurrent-consumers` is greater than 1 and you want all the consumers to be run concurrently, then consider how you configure the work-manager you associate with this JMS adapter:

- If the work-manager is shared with other components (such as other adapters and Jetty) then set the work-manager attribute `max-threads-constraint` greater than or equal to the `concurrent-consumers` setting.
- If the work-manager is not shared (that is, it is dedicated to this inbound JMS adapter only) then set the work-manager attribute `max-threads-constraint` equal to the `concurrent-consumers` setting.

For more information, see:

- [Section , "Creating a Custom Converter Between JMS Messages and Event Types"](#)
- [Section , "work-manager"](#)

## Child Elements

The `concurrent-consumers` component configuration element has no child elements.

## Attributes

The `concurrent-consumers` component has no attributes.

## 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>
```

## 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 Event Processing server JMS.

## Child Elements

The `connection-jndi-name` component configuration element has no child elements.

## Attributes

The `connection-jndi-name` component has no attributes.

## 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>

```

## connection-encrypted-password

Use the `connection-encrypted-password` element to define the encrypted `jms-adapter` password that Oracle Event Processing uses when it acquires a connection to the JMS service provider.

When Oracle Event Processing calls the `createConnection` method on the `javax.jms.ConnectionFactory` to create a connection to the JMS destination (JMS queue or topic), it uses the `connection-user` and `connection-password` or `connection-encrypted-password` element, if configured. Otherwise, Oracle Event Processing uses the `user` and `password` (or `encrypted-password`) elements.

Use either `connection-encrypted-password` or `connection-password` but not both.

For more information, see [Section , "Encrypting Passwords in the JMS Adapter Component Configuration File"](#).

### Child Elements

The `connection-encrypted-password` component configuration element has no child elements.

### Attributes

The `connection-encrypted-password` component has no attributes.

### Example

The following example shows how to use the `connection-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>
  <password>wlevs</password>
  <connection-user>wlevscon</user>
  <encrypted-password>{Salted-3DES}s4YUEvH4Wl2DAjb45iJnrw==</encrypted-password>
</http-pub-sub-adapter>

```

## connection-password

Use the `connection-password` element to define the `jms-adapter` password that Oracle Event Processing uses when it acquires a connection to the JMS service provider.

When Oracle Event Processing calls the `createConnection` method on the `javax.jms.ConnectionFactory` to create a connection to the JMS destination (JMS

queue or topic), it uses the `connection-user` and `connection-password` or `connection-encrypted-password` element, if configured. Otherwise, Oracle Event Processing uses the `user` and `password` (or `encrypted-password`) elements.

Use either `connection-password` or `connection-encrypted-password` but not both.

## Child Elements

The `connection-password` component configuration element has no child elements.

## Attributes

The `connection-password` component has no attributes.

## Example

The following example shows how to use the `connection-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>
  <connection-user>wlevscon</user>
  <connection-password>wlevscon</password>
</http-pub-sub-adapter>
```

## connection-user

Use the `connection-user` element to define the `jms-adapter` user name that Oracle Event Processing uses when it acquires a connection to the JMS service provider.

When Oracle Event Processing calls the `createConnection` method on the `javax.jms.ConnectionFactory` to create a connection to the JMS destination (JMS queue or topic), it uses the `connection-user` and `connection-password` or `connection-encrypted-password` element, if configured. Otherwise, Oracle Event Processing uses the `user` and `password` (or `encrypted-password`) elements.

You can use the `connection-user` and `connection-password` (or `connection-encrypted-password`) settings in applications where one security provider is used for JNDI access and a separate security provider is used for JMS access.

## Child Elements

The `connection-user` component configuration element has no child elements.

## Attributes

The `connection-user` component has no attributes.

## Example

The following example shows how to use the `connection-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>
<connection-user>wlevscon</user>
<connection-password>wlevscon</password>
</http-pub-sub-adapter>

```

## database

Use this element to define a database reference for an EPL processor component.

For more information, see [Chapter 19, "Querying an Event Stream with Oracle EPL"](#).

## Child Elements

The database component configuration element has no child elements.

## Attributes

[Table D-2](#) lists the attributes of the database component configuration element.

**Table D-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 Event Processing server config.xml file.	String	Yes.

## 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>

```

## 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.

## Child Elements

The dataset-name component configuration element has no child elements.

## Attributes

The `dataset-name` component has no attributes.

## 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>
```

## delivery-mode

Use this element to define the delivery mode for a [jms-adapter](#).

Valid values are:

- `persistent` (default)
- `nonpersistent`

## Child Elements

The `delivery-mode` component configuration element has no child elements.

## Attributes

The `delivery-mode` component has no attributes.

## 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>
```

## 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.

## Child Elements

The `destination-jndi-name` component configuration element has no child elements.

## Attributes

The `destination-jndi-name` component has no attributes.

## 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>
```

## 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.

## Child Elements

The `destination-name` component configuration element has no child elements.

## Attributes

The `destination-name` component has no attributes.

## 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>
```

## destination-type

Use this element to define the JMS destination type for a `jms-adapter`. Valid values are `TOPIC` or `QUEUE`. This property must be set to `TOPIC` whenever the `durable-subscription` property is set to `true`.

## Child Elements

The `destination-type` component configuration element has no child elements.

## Attributes

The destination-type component has no attributes.

## Example

The following example shows how to use the destination-type 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>
  <destination-type>TOPIC</destination-name>
  <durable-subscription>true</durable-subscription>
  <durable-subscription-name>JmsDurableSubscription</durable-subscription-name>
  <user>weblogic</user>
  <password>weblogic</password>
  <work-manager>JettyWorkManager</work-manager>
  <concurrent-consumers>1</concurrent-consumers>
  <session-transacted>>false</session-transacted>
</jms-adapter>
```

## diagnostic-profiles

Use this element to define one or more Oracle Event Processing diagnostic profiles.

For more information, see "Monitoring the Throughput and Latency of a Stage or Path in the EPN" in the *Oracle Fusion Middleware Visualizer User's Guide for Oracle Event Processing*.

## Child Elements

The diagnostics-profiles component configuration element supports the following child elements:

- [name](#)
- [profile](#)

## Attributes

The diagnostics-profiles component has no attributes.

## 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.



## 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 Fusion Middleware Visualizer User's Guide for Oracle Event Processing*.

## Child Elements

The direction component configuration has no child elements:

## Attributes

The direction component has no attributes.

## 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>
```

## durable-subscription

Use this element to specify whether the JMS topic subscription of a [jms-adapter](#) is durable, meaning that it can persist even if subscribers become inactive. Valid values are true or false. This property is only valid if [destination-type](#) is set to TOPIC.

## Child Elements

The durable-subscription component configuration element has no child elements.

## Attributes

The durable-subscription component has no attributes.

## Example

The following example shows how to use the durable-subscription 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>
  <destination-type>TOPIC</destination-name>
  <durable-subscription>true</durable-subscription>
  <durable-subscription-name>JmsDurableSubscription</durable-subscription-name>
  <user>weblogic</user>
  <password>weblogic</password>
  <work-manager>JettyWorkManager</work-manager>
  <concurrent-consumers>1</concurrent-consumers>
  <session-transacted>>false</session-transacted>
</jms-adapter>
```

## durable-subscription-name

Use this element to specify the name to uniquely identify a durable subscription of a [jms-adapter](#). A durable subscription can persist even if subscribers become inactive.

## Child Elements

The durable-subscription-name component configuration element has no child elements.

## Attributes

The durable-subscription-name component has no attributes.

## Example

The following example shows how to use the durable-subscription-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>
  <destination-type>TOPIC</destination-name>
  <durable-subscription>true</durable-subscription>
  <durable-subscription-name>JmsDurableSubscription</durable-subscription-name>
  <user>weblogic</user>
  <password>weblogic</password>
  <work-manager>JettyWorkManager</work-manager>
  <concurrent-consumers>1</concurrent-consumers>
  <session-transacted>>false</session-transacted>
</jms-adapter>
```

## 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.

### Child Elements

The duration component configuration element has no child elements.

### Attributes

The duration component has no attributes.

### 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>2010-01-20T05:00:00</start>
    <duration>03:00:00</duration>
  </time-range-offset>
  <batch-size>1</batch-size>
  <batch-time-out>10</batch-time-out>
</record-parameters>
```

## 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 Fusion Middleware Visualizer User's Guide for Oracle Event Processing*.

### Child Elements

The enabled component configuration element has no child elements.

## Attributes

The enabled component has no attributes.

## 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>
```

## 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 Event Processing application is publishing requires user authentication.
- [jms-adapter](#): When Oracle Event Processing acquires the JNDI InitialContext, it uses the [user](#) and [password](#) (or encrypted-password) elements. When Oracle Event Processing calls the createConnection method on the javax.jms.ConnectionFactory to create a connection to the JMS destination (JMS queue or topic), it uses the [connection-user](#) and [connection-password](#) (or [connection-encrypted-password](#) element), if configured. Otherwise, Oracle Event Processing the [user](#) and [password](#) (or encrypted-password) elements.

Use either encrypted-password or [password](#) but not both.

For more information, see [Section , "Encrypting Passwords in the JMS Adapter Component Configuration File"](#).

## Child Elements

The encrypted-password component configuration element has no child elements.

## Attributes

The encrypted-password component has no attributes.

## 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>
```

## end

Use this element to define an end time for a [time-range](#) or [schedule-time-range](#) element.

Express the end time as an XML Schema `dateTime` value of the form:

```
yyyy-mm-ddThh:mm:ss
```

For example, to specify that play back should start on January 20, 2010, at 5:00am and end on January 20, 2010, at 6:00 pm, enter the following:

```
<time-range>
  <start>2010-01-20T05:00:00</start>
  <end>2010-01-20T18:00:00</end>
</time-range>
```

For complete details of the XML Schema `dateTime` format, see

<http://www.w3.org/TR/xmlschema-2/#dateTime-lexical-representation>.

## Child Elements

The end component configuration element has no child elements.

## Attributes

The end component has no attributes.

## 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>2010-01-20T05:00:00</start>
    <end>2010-01-20T18:00:00</end>
  </time-range>
```

```
<batch-size>1</batch-size>
<batch-time-out>10</batch-time-out>
</record-parameters>
```

## 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 Fusion Middleware Visualizer User's Guide for Oracle Event Processing*.

## Child Elements

The end-location component configuration element supports the following child elements:

- [application](#)
- [stage](#)
- [direction](#)

## Attributes

The end-location component has no attributes.

## 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>
```

## event-bean

Use this element to define an event bean component.

For more information, see [Chapter , "Configuring a Java Class as an Event Bean"](#).

## Child Elements

The event-bean component configuration element supports the following child elements:

- [name](#)
- [record-parameters](#)
- [playback-parameters](#)

## Attributes

The event-bean component has no attributes.

## 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.

## event-type

Use the event-type element in the following parent elements:

- [http-pub-sub-adapter](#):
  - Publishing: Optional. For both local and remote HTTP pub-sub adapters for publishing, specify the fully qualified class name of the `JavaBean` event to limit the types of events that are published. Otherwise, all events sent to the HTTP pub-sub adapter are published.
  - Subscribing: Required. For both local and remote HTTP pub-sub adapters for subscribing, specify the fully qualified class name of the `JavaBean` to which incoming messages are mapped. At runtime, Oracle Event Processing uses the incoming key-value pairs in the message to map the message data to the specified event type.

You must register this class in the EPN assembly file as a `wlevs:event-type-repository` element `wlevs:class` child element. For more information, see [Section , "Creating an Oracle Event Processing Event Type as a `JavaBean`"](#).

- [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 Event Processing 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 , "Overview of Oracle Event Processing Event Types"](#).

## Child Elements

The `event-type` component configuration element has no child elements.

## Attributes

The `event-type` component has no attributes.

## 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>
```

## event-type-list

Use this element to define one or more events for record or playback for a component.

## Child Elements

The `event-type-list` component configuration element supports the following child elements:

- [event-type](#)

## Attributes

The `event-type-list` component has no attributes.

## 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>
```

## 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

## Child Elements

The `eviction-policy` component configuration element has no child elements.

## Attributes

The `eviction-policy` component has no attributes.

## 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 >
```

## fail-when-rejected

Use this element to specify whether an `com.bea.wlevs.ede.api.EventProcessingException` should be raised if the event queue is full when the offer timeout expires. If set to `false` or not set at all, then the event is dropped rather than an exception raised. This configuration is only applicable for event queues whose `max-size` value is greater than 0. The default value is `false`.

For more on setting the offer timeout, see ["offer-timeout"](#).

## Child Elements

The `fail-when-rejected` component configuration element has no child elements.

## Attributes

The `fail-when-rejected` component configuration element has no attributes.

## Example

In the following example, the channel is configured to raise an `EventProcessingException` if 15 seconds pass while the event queue is full.

```

<channel>
  <name>QueuedChannel</name>
  <max-size>1000</max-size>
  <max-threads>1</max-threads>
  <offer-timeout>1500000000</offer-timeout>
  <fail-when-rejected>true</fail-when-rejected>
</channel>

```

## group-binding

Edit the component configuration file to add `group-binding` child elements to the `.jms-adapter` element for the JMS inbound adapters.

Add one `group-binding` element for each possible JMS message-selector value.

For more information, see [Section , "bindings \(jms-adapter\)"](#).

## Child Elements

The `group-binding` component configuration element supports the following child elements:

- `param`

## Attributes

[Table D-3](#) lists the attributes of the `group-binding` component configuration element.

**Table D-3 Attributes of the `group-binding` Component Configuration Element**

Attribute	Description	Data Type	Required?
<code>group-id</code>	The name of a cluster element groups child element.	String	Yes.

## Example

The following example shows how to use the `group-binding` element in the component configuration file:

```

<jms-adapter>
  <name>JMSInboundAdapter</name>
  <event-type>StockTick</event-type>
  <jndi-provider-url>t3://ppurich-pc:7001</jndi-provider-url>
  <destination-jndi-name>./Topic1</destination-jndi-name>
  <user>weblogic</user>
  <password>weblogic1</password>
  <work-manager>JettyWorkManager</work-manager>
  <concurrent-consumers>1</concurrent-consumers>
  <session-transacted>true</session-transacted>
  <message-selector>${CONDITION}</message-selector>
  <bindings>
    <group-binding group-id="ActiveActiveGroupBean_group1">
      <param id="CONDITION">acctid > 400</param>
    </group-binding>
    <group-binding group-id="ActiveActiveGroupBean_group2">
      <param id="CONDITION">acctid BETWEEN 301 AND 400</param>
    </group-binding>
    <group-binding group-id="ActiveActiveGroupBean_group3">
      <param id="CONDITION">acctid BETWEEN 201 AND 300</param>
    </group-binding>
    <group-binding group-id="ActiveActiveGroupBean_group4">
      <param id="CONDITION">acctid <= 200</param>
    </group-binding>
  </bindings>

```

```

    </bindings>
</jms-adapter>

```

In this configuration, when the application is deployed to an Oracle Event Processing server with a `cluster` element groups child element that contains `ActiveActiveGroupBean_group1`, then the `CONDITION` parameter is defined as `acctid > 400` and the application processes events whose `acctid` property is greater than 400.

## heartbeat

Use this element to define a new heartbeat timeout for a system-timestamped `channel` component. By default, the timeout value is 100 milliseconds, or 100,000,000 nanoseconds.

For system-timestamped relations or streams, time is dependent upon the arrival of data on the relation or stream data source. Oracle Event Processing 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 the value for this setting. Either the relation or stream is populated by its specified source or Oracle Event Processing generates a heartbeat every `heartbeat` 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.

## Child Elements

The `heartbeat` component configuration element has no child elements.

## Attributes

The `heartbeat` component configuration element has no attributes.

## Example

The following example shows how to use the `heartbeat` element in the component configuration file:

```

<channel>
  <name>MatchOutputChannel</name>
  <max-size>0</max-size>
  <max-threads>0</max-threads>
  <selector>match</selector>
  <heartbeat>10000</heartbeat>
</channel>

```

In the example, the channel's unique identifier is `MatchOutputChannel`.

## http-pub-sub-adapter

Use this element to define an HTTP publish-subscribe server adapter component.

For more information, see [Chapter 12, "Integrating an HTTP Publish-Subscribe Server"](#).

## 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`

## Attributes

The `http-pub-sub-adapter` component configuration element has no attributes.

## 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`.

## 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.

## Child Elements

The `idle-time` component configuration element has no child elements.

## Attributes

The `idle-time` component has no attributes.

## 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 >
```

## inject-parameters

Use this element to configure event injection for a stage in the event processing network.

For more information about event injection, see [Section , "Injecting Events"](#).

## Child Elements

The `inject-parameters` component configuration element supports the following child elements:

- `channel-name`
- `active`

## Attributes

The `inject-parameters` component configuration element has no attributes.

## Example

The component configuration excerpt shown in the following example illustrates how you might configure a processor for event injection. The `inject-parameters` element's `active` child element specifies that injection is on, while the `channel-name` element specifies the HTTP pub-sub channel to which injected elements should be sent.

```
< processor >
  < name > FindCrossRates < / name >
  < inject-parameters >
    < active > true < / active >
    < channel-name > / NonClusteredServer / fx / FindCrossRates / input < / channel-name >
  < / inject-parameters >
  < rules >
    < ! - - Query rules omitted. - - >
  < / rules >
< / processor >
```

## jms-adapter

Use this element to define a JMS adapter component.

For more information, see [Chapter 11, "Integrating the Java Message Service"](#).

### 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`
- `connection-user`
  - One of:
    - `connection-password`
    - `connection-encrypted-password`
- `work-manager`
- `concurrent-consumers`
- `message-selector`
- `session-ack-mode-name`
- `session-transacted`
- `delivery-mode`

### Attributes

The `jms-adapter` component configuration element has no attributes.

### 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`.

## 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 Event Processing server JMS

### Child Elements

The `jndi-factory` component configuration element has no child elements.

### Attributes

The `jndi-factory` component has no attributes.

### 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>

```

## jndi-provider-url

Use this required element to define a JNDI provider URL for a [jms-adapter](#).

### Child Elements

The `jndi-provider-url` component configuration element has no child elements.

### Attributes

The `jndi-provider-url` component has no attributes.

## 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>
```

## 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.

## Child Elements

The `listeners` component configuration element supports the following child elements:

- `work-manager-name`

## Attributes

Table D-4 lists the attributes of the `listeners` component configuration element.

**Table D-4 Attributes of the listeners Component Configuration Element**

Attribute	Description	Data Type	Required?
<code>asynchronous</code>	Execute listeners asynchronously. Valid values are true and false. Default value is false.	Boolean	No.

## 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>
  </cache>
</caching-system>
```



```

        <work-manager-name>JettyWorkManager</work-manager-name>
        <listeners asynchronous="false">
            <work-manager-name>JettyWorkManager</work-manager-name>
        </listeners>
    </cache>
</caching-system>

```

## 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 Fusion Middleware Visualizer User's Guide for Oracle Event Processing*.

## Child Elements

The location component configuration element supports the following child elements:

- [application](#)
- [stage](#)
- [direction](#)

## Attributes

The location component has no attributes.

## 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>

```

## 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 Fusion Middleware Visualizer User's Guide for Oracle Event Processing*.

## Child Elements

The `max-latency` component configuration element supports the following child elements:

- `name`
- `collect-interval`
- `start-location`
- `end-location`

## Attributes

The `max-latency` component has no attributes.

## 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>
```

## 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

## Child Elements

The `max-size` component configuration element has no child elements.

## Attributes

The `max-size` component has no attributes.

## 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>
```

## max-threads

Use this element to define the maximum number of threads that Oracle Event Processing 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.

## Child Elements

The `max-threads` component configuration element has no child elements.

## Attributes

The `max-threads` component has no attributes.

## 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>
```

## 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'
```

### Child Elements

The `message-selector` component configuration element has no child elements.

### Attributes

The `message-selector` component has no attributes.

### 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>
```

### 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.

### Child Elements

The name component configuration element has no child elements:

### Attributes

The name component has no attributes.

## 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`.

## netio

Use this element to define a network input/output port for a component.

---



---

**Note:** When a child of the adapter element, this element is for internal use only.

---



---

## Child Elements

The `netio` component configuration element supports the following child elements:

- [provider-name](#)
- [num-threads](#)
- [accept-backlog](#)

## Attributes

The `netio` component has no attributes.

## 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>
```

## num-threads

Use this element to define the number of threads in a network input/output port for a component.

## Child Elements

The `num-threads` component configuration element has no child elements.

## Attributes

The `num-threads` component has no attributes.

## 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>
```

## offer-timeout

Use this element to specify the amount of time, when an event queue is full and a pending insert (the "offer") is blocked, after which the pending event will be dropped or an exception raised. An exception will be raised when the `fail-when-rejected` value is set to true; otherwise, the event will be dropped. This setting is only applicable for event queues whose `max-size` value is greater than 0. The `offer-timeout` value should be specified as nanoseconds. The default is 60 seconds.

## Child Elements

The `offer-timeout` component configuration element has no child elements:

## Attributes

The `offer-timeout` component has no attributes.

## Example

In the following example, the channel is configured to raise an `EventProcessingException` if 15 seconds pass while the event queue is full.

```
<channel>
  <name>QueuedChannel</name>
  <max-size>1000</max-size>
  <max-threads>1</max-threads>
  <offer-timeout>15000000000</offer-timeout>
  <fail-when-rejected>true</fail-when-rejected>
</channel>
```

## param

Use the `param` element to associate a message selector value with the parameter name specified in the `message-selector` element.

For more information, see [Section , "bindings \(jms-adapter\)"](#).

## Child Elements

The `param` component configuration element has no child elements.

## Attributes

[Table D-5](#) lists the attributes of the `param` component configuration element.

**Table D-5 Attributes of the param Component Configuration Element**

Attribute	Description	Data Type	Required?
id	The parameter name specified in the message-selector.	String	Yes.

## Example

The following example shows how to use the param element in the component configuration file:

```
<jms-adapter>
  <name>JMSInboundAdapter</name>
  <event-type>StockTick</event-type>
  <jndi-provider-url>t3://ppurich-pc:7001</jndi-provider-url>
  <destination-jndi-name>./Topic1</destination-jndi-name>
  <user>weblogic</user>
  <password>weblogic1</password>
  <work-manager>JettyWorkManager</work-manager>
  <concurrent-consumers>1</concurrent-consumers>
  <session-transacted>>true</session-transacted>
  <message-selector>${CONDITION}</message-selector>
  <bindings>
    <group-binding group-id="ActiveActiveGroupBean_group1">
      <param id="CONDITION">acctid > 400</param>
    </group-binding>
    <group-binding group-id="ActiveActiveGroupBean_group2">
      <param id="CONDITION">acctid BETWEEN 301 AND 400</param>
    </group-binding>
    <group-binding group-id="ActiveActiveGroupBean_group3">
      <param id="CONDITION">acctid BETWEEN 201 AND 300</param>
    </group-binding>
    <group-binding group-id="ActiveActiveGroupBean_group4">
      <param id="CONDITION">acctid <= 200</param>
    </group-binding>
  </bindings>
</jms-adapter>
```

In this configuration, when the application is deployed to an Oracle Event Processing server with a cluster element groups child element that contains ActiveActiveGroupBean\_group1, then the CONDITION parameter is defined as acctid > 400 and the application processes events whose acctid property is greater than 400.

## parameter

Use this element to define a name/value parameter for a component.

### Child Elements

The parameter component configuration element supports the following child elements:

- [name](#)
- [value](#)

### Attributes

The parameter component has no attributes.

## 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>
```

## 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 Fusion Middleware CQL Language Reference for Oracle Event Processing*
- "Parameterized Queries" in the *Oracle Fusion Middleware EPL Language Reference for Oracle Event Processing*

## Child Elements

The `params` component configuration element has no child elements.

## Attributes

[Table D-6](#) lists the attributes of the `params` component configuration element.

**Table D-6 Attributes of the `params` Component Configuration Element**

Attribute	Description	Data Type	Required?
<code>id</code>	Unique identifier for this <code>params</code> element.	String	No.

## 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>
  </record-parameters>
</processor>
```



```

    <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>

```

## partition-order-capacity

Use this element to define the maximum capacity of a query partition when the ordering-constraint attribute is set to `PARTITION_ORDERED`. Set this on a [channel](#) component. Consider setting this element's value when you've configured a query processor for parallel execution, and when the query's ordering-constraint attribute is set to `PARTITION_ORDERED`.

For more information, including best practices and information on the locations where this value can be set (including their precedence), see "Using partition-order-capacity with Partitioning Queries" in [Chapter 17, "Querying an Event Stream with Oracle CQL"](#).

To have the capacity value apply at a larger scope, you can set it in the server configuration file. For more information, see "partition-order-capacity" in [Appendix F, "Schema Reference: Server Configuration wlevs\\_server\\_config.xsd"](#).

### Child Elements

The `partition-order-capacity` component configuration element has no child elements.

### Attributes

The `partition-order-capacity` component configuration element has no attributes.

### Example

The following example shows how to use the `partition-order-capacity` element in the component configuration file:

```

<channel>
  <name>MatchOutputChannel</name>
  <max-size>0</max-size>
  <max-threads>0</max-threads>
  <selector>match</selector>
  <partition-order-capacity>20</partition-order-capacity>
</channel>

```

## 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 Event Processing application is publishing requires user authentication.
- [jms-adapter](#): When Oracle Event Processing acquires the JNDI `InitialContext`, it uses the `user` and `password` (or [encrypted-password](#)) elements. When Oracle Event Processing calls the `createConnection` method on the `javax.jms.ConnectionFactory` to create a connection to the JMS destination (JMS queue or topic), it uses the [connection-user](#) and [connection-password](#) (or [connection-encrypted-password](#) element), if configured. Otherwise, Oracle Event Processing the `user` and `password` (or [encrypted-password](#)) elements.

Use either [encrypted-password](#) or `password` but not both.

### Child Elements

The password component configuration element has no child elements.

### Attributes

The password component has no attributes.

### 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>
```

## playback-parameters

Use this element to define event playback parameters for a component.

For more information, see [Chapter 20, "Configuring Event Record and Playback"](#).

### 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`

## Attributes

The `playback-parameters` component has no attributes.

## 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>
```

## 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.

## Child Elements

The `playback-speed` component configuration element has no child elements.

## Attributes

The `playback-speed` component has no attributes.

## 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>
</playback-parameters>
```

```
    <parameter>
  </store-policy-parameters>
  <time-range-offset>
    <start>2010-01-20T05:00:00</start>
    <duration>03:00</duration>
  </time-range-offset>
  <playback-speed>100</playback-speed>
</playback-parameters>
```

## processor (EPL)

Use this element to define an Oracle CQL or EPL processor component.

For more information, see [Chapter 19, "Querying an Event Stream with Oracle EPL"](#).

For information on the processor element for Oracle CQL processors, see [processor \(Oracle CQL\)](#).

## Child Elements

The processor component configuration element supports the following child elements:

- [name](#)
- [record-parameters](#)
- [playback-parameters](#)
- [rules](#)
- [database](#)
- [bindings \(processor\)](#)

## Attributes

The processor component has no attributes.

## 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 processor1.

## processor (Oracle CQL)

Use this element to define an Oracle CQL processor component.

For more information, see [Chapter 17, "Querying an Event Stream with Oracle CQL"](#).

For information on the processor element for EPL processors, see [processor \(EPL\)](#).

### Child Elements

The processor component configuration element supports the following child elements:

- [name](#)
- [record-parameters](#)
- [playback-parameters](#)
- [rules](#)
- [bindings \(processor\)](#)

### Attributes

The processor component has no attributes.

### 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>
  </rules>
</processor>

```

```

<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>
<query id="MarketRule"><![CDATA[
    SELECT symbol, AVG(price) AS average, :1 AS market
    FROM StockTick [RANGE 5 SECONDS]
    WHERE symbol = :2
]]></query>
</rules>
<bindings>
  <binding id="MarketRule">
    <params id="nasORCL">NASDAQ, ORCL</params>
    <params id="nyJPM">NYSE, JPM</params>
    <params id="nyWFC">NYSE, WFC</params>
  </binding>
</bindings>
</processor>

```

In the example, the processor's unique identifier is `cqlProcessor`.

## 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 Fusion Middleware Visualizer User's Guide for Oracle Event Processing*.

## Child Elements

The profile component configuration element supports the following child elements:

- `name`
- `enabled`
- `start-stage`
- `max-latency`
- `average-latency`
- `throughput`

## Attributes

The profile component has no attributes.

## 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>
<threshold>
    <amount>100</amount>
    <unit>MILLISECONDS</unit>
</threshold>
</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>

```

## 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 Event Processing 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 Event Processing server instance.

When configuring the Oracle RDBMS-based provider, you are required to specify this element.

This may be left blank to configure to use the default Berkeley database provider.

## Child Elements

The `provider-name` component configuration element has no child elements.

## Attributes

The `provider-name` component has no attributes.

## 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>
```

## query

Use this element to define an Oracle CQL query for a component.

For more information, see [Chapter 17, "Querying an Event Stream with Oracle CQL"](#).

## Child Elements

The `query` component configuration element has no child elements.

## Attributes

[Table D-7](#) lists the attributes of the `query` component configuration element.

**Table D-7 Attributes of the query Component Configuration Element**

Attribute	Description	Data Type	Required?
<code>id</code>	Unique identifier for this query.	String	Yes.
<code>active</code>	Execute this query when the application is deployed and run. Valid values are <code>true</code> and <code>false</code> . Default value is <code>false</code> .	Boolean	No.
<code>ordering-constraint</code>	Enable or disable parallel query execution, through which events can be processed in parallel rather than serially. The attribute supports the following three values: <ul style="list-style-type: none"> <li><code>ORDERED</code> means that the query must handle events serially. This is the default behavior.</li> <li><code>UNORDERED</code> means that, whenever possible, the CQL processor will execute in parallel on multiple threads to process the events.</li> <li><code>PARTITION_ORDERED</code> means that when the query is partitioning events, ensure total order within a partition and (if possible) disregard order across partitions.</li> </ul> For more information see "Using the <code>ordering-constraint</code> Attribute" in <a href="#">Chapter 17, "Querying an Event Stream with Oracle CQL"</a> .	String	No.



**Table D-7 (Cont.) Attributes of the query Component Configuration Element**

Attribute	Description	Data Type	Required?
partition-expression	The partition expression (used in the CQL code) that should be the basis for relaxing the cross-partition ordering constraint.  For more information see "Using the ordering-constraint Attribute" in <a href="#">Chapter 17, "Querying an Event Stream with Oracle CQL"</a> .	String	No.

## 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>
```

## record-parameters

Use this element to define event record parameters for a component.

For more information, see [Chapter 20, "Configuring Event Record and Playback"](#).

## 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](#)

## Attributes

The `record-parameters` component has no attributes.

## 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>
```

## repeat

Use this element to define whether or not to repeat [playback-parameters](#).

Valid values are:

- `true`
- `false`

## Child Elements

The `repeat` component configuration element has no child elements.

## Attributes

The `repeat` component has no attributes.

## 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>2010-01-20T05:00:00</start>
  <duration>03:00:00</duration>
</time-range-offset>
<repeat>true</repeat>
</playback-parameters>

```

## rule

Use this element to define an EPL rule for a component.

This element is applicable only in a [rules](#) element.

## Child Elements

The rule component configuration element has no child elements.

## Attributes

[Table D-8](#) lists the attributes of the rule component configuration element.

**Table D-8 Attributes of the rule Component Configuration Element**

Attribute	Description	Data Type	Required?
id	Unique identifier for this rule.	String	Yes.
active	Execute this rule when the application is deployed and run. Valid values are true and false. Default value is false.	Boolean	No.
ordering-constraint	Enable or disable parallel query execution, through which events can be processed in parallel rather than serially. The attribute supports the following three values: <ul style="list-style-type: none"> <li>ORDERED means that the query must handle events serially. This is the default behavior.</li> <li>UNORDERED means that, whenever possible, the CQL processor will execute in parallel on multiple threads to process the events.</li> <li>PARTITION_ORDERED means that when the query is partitioning events, ensure total order within a partition and (if possible) disregard order across partitions.</li> </ul> For more information see "Using the ordering-constraint Attribute" in <a href="#">Chapter 17, "Querying an Event Stream with Oracle CQL"</a> .	String	No.
partition-expression	The partition expression (used in the CQL code) that should be the basis for relaxing the cross-partition ordering constraint. For more information see "Using the ordering-constraint Attribute" in <a href="#">Chapter 17, "Querying an Event Stream with Oracle CQL"</a> .	String	No.

## 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>
```

## 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 \(EPL\)](#).

## Child Elements

The rules component configuration element supports the following child elements:

- [rule](#)
- [query](#)
- [view](#)

## Attributes

The rules component has no attributes.

## 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>

```

## 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.

### Child Elements

The `schedule-time-range` component configuration element supports the following child elements:

- `start`
- `end`

### Attributes

The `schedule-time-range` component has no attributes.

### 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>2010-01-20T05:00:00</start>
    <end>2010-01-20T18:00:00</end>
  </schedule-time-range>
  <batch-size>1</batch-size>
  <batch-time-out>10</batch-time-out>
</record-parameters>

```

## 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.

## Child Elements

The `schedule-time-range-offset` component configuration element supports the following child elements:

- [start](#)
- [duration](#)

## Attributes

The `schedule-time-range-offset` component has no attributes.

## 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>2010-01-20T05:00:00</start>
    <duration>03:00:00</duration>
  </schedule-time-range-offset>
  <batch-size>1</batch-size>
  <batch-time-out>10</batch-time-out>
</record-parameters>
```

## selector

Use this element to specify which up-stream Oracle CQL processor queries are permitted to output their results to a downstream [channel](#).

[Figure D-1](#) shows an EPN with channel `filteredStream` connected to up-stream Oracle CQL processor `filteredFanoutProcessor`.

**Figure D-1 EPN With Oracle CQL Processor and Down-Stream Channel**



[Example D-12](#) shows the queries configured for the Oracle CQL processor.

### **Example D-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 D-12](#) shows, then, by default, all query results are output to the processor's out-bound channel (filteredStream in [Figure D-1](#)). Optionally, in the component configuration source, you can use the channel element selector child element to specify a space-delimited list of one or more Oracle CQL query names that may output their results to the channel as [Example D-13](#) shows. In this example, query results for query Yr3Sector and Yr2Sector are output to filteredStream but not query results for query Yr1Sector.

#### **Example D-13 Using selector to Control Which Query Results are Output**

```

<channel>
  <name>filteredStream</name>
  <selector>Yr3Sector Yr2Sector</selector>
</channel>

```

You may configure a channel element with a selector before creating the queries in the upstream processor. In this case, you must specify query names that match the names in the selector.

---

**Note:** The selector attribute is only applicable if the up-stream node is an Oracle CQL processor. For more information, see [Chapter 17, "Querying an Event Stream with Oracle CQL"](#).

---

## Child Elements

The selector component configuration element has no child elements.

## Attributes

The selector component has no attributes.

## 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>

```

## server-context-path

Required. For each *local* [http-pub-sub-adapter](#) for publishing, specify the value of the Oracle Event Processing server `config.xml` file element `http-pubsub` child element `path` of the local HTTP pub-sub server associated with the Oracle Event Processing instance hosting the current Oracle Event Processing application.

Default: `/pubsub`.

If you have created a new local HTTP pub-sub server, or changed the default configuration, then specify the appropriate `path` child element value.

---



---

**Note:** Do not specify this option for a remote HTTP pub-sub adapter.

---



---

## Child Elements

The `server-context-path` component configuration element has no child elements.

## Attributes

The `server-context-path` component has no attributes.

## 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>

```

## server-url

Required. For each *remote* [http-pub-sub-adapter](#) for publishing or subscribing, specify the URL of the remote HTTP pub-sub server to which the Oracle Event Processing application will publish. The remote HTTP pub-sub server could be



another instance of Oracle Event Processing, or a WebLogic Server instance, or it could be any third-party HTTP pub-sub server. For example:

```
http://myhost.com:9102/pubsub
```

---



---

**Note:** Do not specify this option for a local HTTP pub-sub adapter.

---



---

## Child Elements

The `server-url` component configuration element has no child elements.

## Attributes

The `server-url` component has no attributes.

## 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`.

## 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.

Default: `AUTO_ACKNOWLEDGE`.

## Child Elements

The `session-ack-mode-name` component configuration element has no child elements.

## Attributes

The `session-ack-mode-name` component has no attributes.

## 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>
```

## session-transacted

Use this element to define whether or not a session is transacted for both an inbound or outbound [jms-adapter](#).

Valid values are:

- true
- false

## Child Elements

The `session-transacted` component configuration element has no child elements.

## Attributes

The `session-transacted` component has no attributes.

## 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>
```

## 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).

## Child Elements

The `stage` component configuration has no child elements:

## Attributes

The stage component has no attributes.

## 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>
```

## start

Use this element to define a start time for a [time-range](#), [time-range-offset](#), or [schedule-time-range-offset](#) element.

Express the start time as an XML Schema `dateTime` value of the form:

```
yyyy-mm-ddThh:mm:ss
```

For example, to specify that play back should start on January 20, 2010, at 5:00am and end on January 20, 2010, at 6:00 pm, enter the following:

```
<time-range>
  <start>2010-01-20T05:00:00</start>
  <end>2010-01-20T18:00:00</end>
</time-range>
```

For complete details of the XML Schema `dateTime` format, see <http://www.w3.org/TR/xmlschema-2/#dateTime-lexical-representation>.

## Child Elements

The `start` component configuration element has no child elements.

## Attributes

The `start` component has no attributes.

## 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>2010-01-20T05:00:00</start>
    <end>2010-01-20T18:00:00</end>
  </time-range>
  <batch-size>1</batch-size>
  <batch-time-out>10</batch-time-out>
</record-parameters>
```

## 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 Fusion Middleware Visualizer User's Guide for Oracle Event Processing*.

## Child Elements

The `start-location` component configuration element supports the following child elements:

- [application](#)
- [stage](#)
- [direction](#)

## Attributes

The `start-location` component has no attributes.

## 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>
    </max-latency>
  </profile>
</diagnostic-profiles>
```

```

    </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>

```

## 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 "Monitoring the Throughput and Latency of a Stage or Path in the EPN" in the *Oracle Fusion Middleware Visualizer User's Guide for Oracle Event Processing*.

## Child Elements

The start-stage component configuration element has no child elements.

## Attributes

The start-stage component has no attributes.

## 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>

```

## store-policy-parameters

Use this element to define one or more store policy parameter, specific to the event store provider.

### Child Elements

The `store-policy-parameter` component configuration element supports the following child elements:

- [parameter](#)

### Attributes

The `store-policy-parameter` component has no attributes.

### 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>
```

## stream

Use this element to define a stream component.

---

---

**Note:** The `stream` component is deprecated in 11g Release 1 (11.1.1). Use the `channel` element instead.

---

---

### Child Elements

The `stream` component configuration element supports the following child elements:

- [name](#)
- [record-parameters](#)
- [playback-parameters](#)
- [max-size](#)
- [max-threads](#)

### Attributes

The `stream` component has no attributes.

## 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`.

## 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 11g Release 1 (11.1.1).

---



---

## Child Elements

The `symbol` component configuration has no child elements:

## Attributes

The `symbol` component has no attributes.

## 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>
```

## symbols

Use this element to define one or more `symbol` elements for a component.

---



---

**Note:** The `symbol` component is deprecated in 11g Release 1 (11.1.1).

---



---

## Child Elements

The `symbols` component configuration element supports the following child elements:

- [symbol](#)

## Attributes

The `symbols` component has no attributes.

## 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>
```

## threshold

Use this element to define the threshold above which Oracle Event Processing 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 Fusion Middleware Visualizer User's Guide for Oracle Event Processing*.

## Child Elements

The threshold component configuration element supports the following child elements:

- `amount`
- `unit`

## Attributes

The threshold component has no attributes.

## Example

The following example shows how to use the `threshold` 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>
      <threshold>
        <amount>100</amount>
        <unit>MILLISECONDS</unit>
      </threshold>
    </average-latency>
  </profile>
</diagnostic-profiles>
```



```

        </average-latency>
    </profile>
</diagnostic-profiles>

```

## 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 Fusion Middleware Visualizer User's Guide for Oracle Event Processing*.

## Child Elements

The throughput component configuration element supports the following child elements:

- `name`
- `throughput-interval`
- `average-interval`
- `location`

## Attributes

The throughput component has no attributes.

## 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>

```

## 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 Fusion Middleware Visualizer User's Guide for Oracle Event Processing*.

## Child Elements

The throughput-interval component configuration element supports the following child elements:

- [amount](#)
- [unit](#)

## Attributes

The throughput-interval component has no attributes.

## 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>
```

## time-range

Use this element to define a filter that Oracle Event Processing 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 20, "Configuring Event Record and Playback"](#).

## Child Elements

The time-range component configuration element supports the following child elements:

- [start](#)

- [end](#)

## Attributes

The time-range component has no attributes.

## 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>2010-01-20T05:00:00</start>
    <end>2010-01-20T18:00:00</end>
  </time-range>
  <batch-size>1</batch-size>
  <batch-time-out>10</batch-time-out>
</record-parameters>
```

## time-range-offset

Use this element to define a filter that Oracle Event Processing 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 20, "Configuring Event Record and Playback"](#).

## Child Elements

The time-range-offset component configuration element supports the following child elements:

- [start](#)
- [duration](#)

## Attributes

The time-range-offset component has no attributes.

## 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>2010-01-20T05:00:00</start>
        <duration>03:00:00</duration>
    </time-range-offset>
    <batch-size>1</batch-size>
    <batch-time-out>10</batch-time-out>
</record-parameters>
```

## 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 , "Configuring an Oracle Event Processing Local Caching System and Cache"](#).

## Child Elements

The `time-to-live` component configuration element has no child elements.

## Attributes

The `time-to-live` component has no attributes.

## 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>
```

## trace-parameters

Use this element to configure event tracing for a stage in the event processing network.

For more information about event tracing, see [Section , "Tracing Events"](#).

## Child Elements

The `trace-parameters` component configuration element supports the following child elements:

- `channel-name`
- `active`

## Attributes

The `trace-parameters` component configuration element has no attributes.

## Example

The component configuration excerpt shown in the following example illustrates how you might configure a processor for event tracing. The `trace-parameters` element's `active` child element specifies that tracing is on, while the `channel-name` element specifies the HTTP pub-sub channel to which traced elements should be sent.

```
<processor>
  <name>FindCrossRates</name>
  <trace-parameters>
    <active>true</active>
    <channel-name>/NonClusteredServer/fx/FindCrossRates/output</channel-name>
  </trace-parameters>
  <rules>
    <!-- Query rules omitted. -->
  </rules>
</processor>
```

## unit

Use this element to define the duration units of `amount` element.

Valid values are:

- NANoseconds
- MICROseconds
- MILLIseconds
- SECONDS
- MINUTES
- HOURS
- DAYS

## Child Elements

The `unit` component configuration has no child elements:

## Attributes

The `unit` component has no attributes.

## 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>
```

## 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 Event Processing application is publishing requires user authentication.
- [jms-adapter](#): When Oracle Event Processing acquires the JNDI `InitialContext`, it uses the `user` and [password](#) (or [encrypted-password](#)) elements. When Oracle Event Processing calls the `createConnection` method on the `javax.jms.ConnectionFactory` to create a connection to the JMS destination (JMS queue or topic), it uses the [connection-user](#) and [connection-password](#) (or [connection-encrypted-password](#) element), if configured. Otherwise, Oracle Event Processing uses the `user` and [password](#) (or [encrypted-password](#)) elements.

## Child Elements

The `user` component configuration element has no child elements.

## Attributes

The `user` component has no attributes.

## 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>

```

## value

Use this element to define the value of a [parameter](#) element.

## Child Elements

The value component configuration element has no child elements.

## Attributes

The value component has no attributes.

## 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>

```

## view

Use this element to define an Oracle CQL view for a component.

For more information, see [Chapter 17, "Querying an Event Stream with Oracle CQL"](#).

## Child Elements

The view component configuration element has no child elements.

## Attributes

[Table D-9](#) lists the attributes of the view component configuration element.

**Table D-9** Attributes of the view Component Configuration Element

Attribute	Description	Data Type	Required?
id	Unique identifier for this query.	String	Yes.

**Table D-9 (Cont.) Attributes of the view Component Configuration Element**

Attribute	Description	Data Type	Required?
active	Execute this query when the application is deployed and run. Valid values are true and false. Default value is false.	Boolean	No.
ordering-constraint	Enable or disable parallel query execution, through which events can be processed in parallel rather than serially. The attribute supports one of the following three values: <ul style="list-style-type: none"> <li>▪ ORDERED means that the query must handle events serially. This is the default behavior.</li> <li>▪ UNORDERED means that, whenever possible, the CQL processor will execute in parallel on multiple threads to process the events.</li> <li>▪ PARTITION_ORDERED means that when the query is partitioning events, ensure total order within a partition and (if possible) disregard order across partitions.</li> </ul> For more information see "Using the ordering-constraint Attribute" in <a href="#">Chapter 17, "Querying an Event Stream with Oracle CQL"</a> .	String	No.
partition-expression	The partition expression (used in the CQL code) that should be the basis for relaxing the cross-partition ordering constraint. For more information see "Using the ordering-constraint Attribute" in <a href="#">Chapter 17, "Querying an Event Stream with Oracle CQL"</a> .	String	No.
schema	Space delimited list of stream elements used in the view.	String of space delimited tokens.	No.

## 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>

```

## 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 Event Processing server `config.xml` file `work-manager` element name child element. The default value is the work manager configured for the application itself.

For more information, see [Section , "work-manager"](#).

### Child Elements

The `work-manager` component configuration element has no child elements:

### Attributes

The `work-manager` component has no attributes.

### 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>

```

## 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 Event Processing server `config.xml` file `work-manager` element name child element. The default value is the work manager configured for the application itself.

### Child Elements

The `work-manager-name` component configuration element has no child elements:

## Attributes

The `work-manager-name` component has no attributes.

## 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>
```

## 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.

## 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`

## Attributes

The `write-behind` component has no attributes.

## 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>
    </write-behind>
  </cache>
</caching-system>
```

```

        <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>

```

## 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.

### Child Elements

The write-none component configuration element has no child elements.

### Attributes

The write-none component has no attributes.

### 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>

```

## 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.

### Child Elements

The write-through component configuration element has no child elements.

### Attributes

The write-through component has no attributes.

## 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 >
```

---

---

## Schema Reference: Deployment deployment.xsd

This appendix provides a reference to the elements of the `deployment.xsd` schema, the schema behind the XML with which you configure Oracle Event Processing application deployment.

This appendix includes the following sections:

- [Overview of the Oracle Event Processing Deployment Elements](#)
- [wlevs:deployment](#)

### Overview of the Oracle Event Processing Deployment Elements

Oracle Event Processing 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.

### Element Hierarchy

The Oracle Event Processing component configuration elements are organized into the following hierarchy:

beans

Standard Spring and OSGi elements such as `bean`, `osgi-service`, and so on.

### Example of an Oracle Event Processing Deployment Configuration File

The following sample deployment configuration file from the `fx` application shows how to use many of the Oracle Event Processing 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"/>
```

&lt;/beans&gt;

## wlevs:deployment

Use this element to declare an adapter component to the Spring application context.

### Child Elements

The `wlevs:deployment` deployment element has no child elements:

### 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?
<code>id</code>	Unique identifier for this deployed application.	String	Yes.
<code>depends-on</code>	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.
<code>location</code>	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 <code>DOMAIN_DIR</code> domain directory.  For example: <code>location="file:applications/simpleApp/simpleApp.jar"</code>  Specifies that the bundle <code>simpleApp.jar</code> , located in the <code>DOMAIN_DIR/applications/simpleApp</code> directory, is to be deployed to Oracle Event Processing server.	String	No.
<code>state</code>	Specifies the state that the bundle should be in once it is deployed to the Oracle Event Processing server. The value of this attribute must be one of the following: <ul style="list-style-type: none"> <li><code>start</code>: Install and start the bundle so that it immediately begins taking client requests.</li> <li><code>install</code>: Install the bundle, but do not start it.</li> <li><code>update</code>: Update an existing bundle.</li> </ul> Default value: <code>start</code> .	String	No.

### Example

The following example shows how to use the `wlevs:deployment` element in the deployment file:

```
<?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>
```







---

---

## Schema Reference: Server Configuration wlevs\_server\_config.xsd

This appendix provides a reference to elements of the `wlevs_server_config.xsd` schema, the schema behind XML you use to configure Oracle Event Processing server attributes and services such as logging, Oracle Continuous Query Language (CQL), Secure Sockets Layer (SSL), Java Management Extensions (JMX), HTTP Publish-Subscribe, and more.

This appendix includes the following sections:

- [Overview of the Oracle Event Processing Server Configuration Elements](#)
- [auth-constraint](#)
- [bdb-config](#)
- [channels](#)
- [channel-constraints](#)
- [channel-resource-collection](#)
- [cluster](#)
- [connection-pool-params](#)
- [cql](#)
- [data-source](#)
- [data-source-params](#)
- [driver-params](#)
- [domain](#)
- [debug](#)
- [event-store](#)
- [exported-jndi-context](#)
- [http-pubsub](#)
- [jetty](#)
- [jetty-web-app](#)
- [jmx](#)
- [jndi-context](#)
- [log-file](#)

- [log-stdout](#)
- [logging-service](#)
- [message-filters](#)
- [name](#)
- [netio](#)
- [netio-client](#)
- [partition-order-capacity](#)
- [path](#)
- [pubsub-bean](#)
- [rdbms-event-store-provider](#)
- [rmi](#)
- [scheduler](#)
- [server-config](#)
- [services](#)
- [show-detail-error-message](#)
- [ssl](#)
- [timeout-seconds](#)
- [transaction-manager](#)
- [use-secure-connections](#)
- [weblogic-instances](#)
- [weblogic-jta-gateway](#)
- [weblogic-rmi-client](#)
- [work-manager](#)
- [xa-params](#)

## Overview of the Oracle Event Processing Server Configuration Elements

Oracle Event Processing provides a number of server configuration elements that you use to configure Oracle Event Processing server-specific attributes and services.

### Element Hierarchy

The top-level Oracle Event Processing 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
- bdb-config
- rdbms-event-store-provider
- ssl
- weblogic-rmi-client
- weblogic-jta-gateway
- use-secure-connections
- show-detail-error-message
- cql

## Example of an Oracle Event Processing Server Configuration File

The following sample Oracle Event Processing server configuration file from the HelloWorld application shows how to use many of the Oracle Event Processing elements:

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

```

```

        <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>

</nl:config>

```

## auth-constraint

Use this element to configure an authorization constraint for a [channel-constraints](#) element.

For more information on channels, see [channels](#).

## 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.

**Table F-1 (Cont.) Child Elements of: auth-constraint**

XML Tag	Type	Description
role-name	string	A valid role name. "Users, Groups, and Roles" in the <i>Oracle Fusion Middleware Administrator's Guide for Oracle Event Processing</i>

## Attributes

The auth-constraint server configuration element has no attributes.

## Example

The following example shows how to use the auth-constraint element in the Oracle Event Processing 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>
```

## bdb-config

Use this element to configure the default event store provider that uses a Berkeley database instance.

Optionally, you may configure Oracle Event Processing server to use a relational database instance as the event store provider as [Section , "rdbms-event-store-provider"](#) describes.

## Child Elements

The bdb-config server configuration element supports the child elements that [Table F-2](#) lists

**Table F-2 Child Elements of: bdb-config**

XML Tag	Type	Description
db-env-path	string	Specifies the subdirectory in which Oracle Event Processing server creates Berkeley database instances relative to the <i>DOMAIN_DIR/servername/config</i> directory of your server, where <i>DOMAIN_DIR</i> refers to the domain directory, such as <i>/oracle_cep/user_projects/domains/myDomain</i> and <i>servername</i> refers to the name of your server, such as <i>defaultserver</i> . Default: bdb

**Table F–2 (Cont.) Child Elements of: bdb-config**

XML Tag	Type	Description
cache-size	long	<p>Specifies the amount of memory, in bytes, available for Berkeley database cache entries. You can adjust the cache size to tune Berkeley database performance.</p> <p>For more information, see:</p> <ul style="list-style-type: none"> <li>▪ <a href="http://www.oracle.com/technology/documentation/berkeley-db/je/GettingStartedGuide/cachesize.html">http://www.oracle.com/technology/documentation/berkeley-db/je/GettingStartedGuide/cachesize.html</a>.</li> <li>▪ <a href="http://www.oracle.com/technology/documentation/berkeley-db/je/java/com/sleepycat/je/EnvironmentMutableConfig.html#setCacheSize(long)">http://www.oracle.com/technology/documentation/berkeley-db/je/java/com/sleepycat/je/EnvironmentMutableConfig.html#setCacheSize(long)</a></li> </ul> <p>Default: <code>je.maxMemoryPercent * JVM maximum memory</code></p>

## Attributes

The `bdb-config` server configuration element has no attributes.

## Example

The following example shows how to use the `bdb-config` element in the Oracle Event Processing server configuration file:

```
<bdb-config>
  <db-env-path>bdb</db-env-path>
  <cache-size>1000</cache-size>
</bdb-config>
```

## 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

## 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.

## Attributes

The `channels` server configuration element has no attributes.

## Example

The following example shows how to use the `channels` element in the Oracle Event Processing 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>
```

## channel-constraints

Use this element to configure one or more channel constraints for a [pubsub-bean](#) element.

For more information on channels, see [channels](#).

## 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](#)

## Attributes

The `channel-constraints` server configuration element has no attributes.

## Example

The following example shows how to use the `channel-constraints` element in the Oracle Event Processing 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>
          </element>
        </channel-resource-collection>
      </element>
    </channel-constraints>
  </pub-sub-bean>
</http-pubsub>
```



```

        <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>

```

## 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](#).

### Child Elements

The channel-resource-collection server configuration element contains zero or more element child elements that support the child elements that [Table F-3](#) lists

**Table F-3** 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 <code>element</code> child element with a string value.
channel-patterns	string	Specifies a channel pattern. This element contains an <code>element</code> child element with a string value.
channel-operations	string	Specifies the operation to channel, validate values include: <ul style="list-style-type: none"> <li>▪ create</li> <li>▪ delete</li> <li>▪ subscribe</li> <li>▪ publish</li> </ul> This element contains an <code>element</code> child element with a string value.

### Attributes

The channel-resource-collection server configuration element has no attributes.

### Example

The following example shows how to use the channel-resource-collection element in the Oracle Event Processing 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>

```

## cluster

Use this element to configure a cluster component in the Oracle Event Processing server.

For more information, see "Administrating Multi-Server Domains With Oracle Event Processing Native Clustering" in the *Oracle Fusion Middleware Administrator's Guide for Oracle Event Processing*.

## Child Elements

The `cluster` server configuration element supports the child elements that [Table F-4](#) lists.

**Table F-4** Child Elements of: `cluster`

XML Tag	Type	Description
<code>name</code>	string	The name of this cluster. For more information, see <a href="#">name</a> .
<code>server-name</code>	string	Specifies a unique name for the server. Oracle Event Processing Visualizer uses the value of this element when it displays the server in its console. Default value: <ul style="list-style-type: none"> <li>▪ Oracle Event Processing native clustering: <code>WLEvServer-identity</code> where <i>identity</i> is the value of the <code>identity</code> element.</li> <li>▪ Oracle Coherence: <code>WLEvServer-identity</code> where <i>identity</i> is the member ID as determined by Oracle Coherence.</li> </ul>
<code>server-host-name</code>	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.

**Table F-4 (Cont.) Child Elements of: cluster**

XML Tag	Type	Description
multicast-address	string	<p>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 <code>multicast-address</code> element and Oracle Event Processing automatically assigns a multicast address to the multi-server domain based on the computer's IP address.</p> <p>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 <code>239.255.X.X</code>, which is what the auto-assigned multicast address is based on.</p> <p>All the Oracle Event Processing servers using this <code>multicast-address</code> must be on the same subnet.</p> <p>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 multicast.</p>
multicast-interface	string	<p>The name of the interface that the multicast address should be bound to. This can be one of:</p> <ul style="list-style-type: none"> <li>▪ Simple name, such as <code>eth0</code>.</li> <li>▪ IP address to which the NIC is bound, such as <code>192.168.1.2</code>.</li> <li>▪ IP address and network mask to which the NIC is bound separated by a <code>/</code>, such as <code>192.68.1.2/255.255.255.0</code>.</li> </ul>
multicast-port	int	Specifies the port used for multicast traffic. Default value is 9100.
identity	string	<p>Applicable only to Oracle Event Processing native clustering: specifies the server's identity and must be an integer between 1 and <code>INT_MAX</code>. Oracle Event Processing 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.</p> <p>Not applicable to Oracle Coherence.</p>
enabled	See Description	<p>Specifies whether or not the cluster is enabled. Valid values:</p> <ul style="list-style-type: none"> <li>▪ <code>coherence</code></li> <li>▪ <code>evs4j</code></li> <li>▪ <code>true</code>: cluster is enabled (Oracle Coherence mode)</li> <li>▪ <code>false</code>: cluster is not enabled (default).</li> </ul>
security	See Description	<p>Specifies the type of security for this cluster. Valid values:</p> <ul style="list-style-type: none"> <li>▪ <code>none</code>—Default value. Specifies that no security is configured for the multi-server domain.</li> <li>▪ <code>encrypt</code>—Specifies that multi-server messages should be encrypted.</li> </ul>
groups	string	Specifies a comma-separated list of the names of the groups this cluster belongs to. For more information, see "Groups" in the <i>Oracle Fusion Middleware Administrator's Guide for Oracle Event Processing</i> .
operation-timeout	int	Specifies, in milliseconds, the timeout for point-to-point HTTP multi-server requests. Default value is 30000.

## Attributes

The cluster server configuration element has no attributes.

## Example

The following example shows how to use the `cluster` element in the Oracle Event Processing 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`.

## connection-pool-params

Use this element to specify connection pool-related [data-source](#) parameters.

### Child Elements

The `connection-pool-params` server configuration element supports the child elements that [Table F-5](#) lists.

**Table F-5 Child Elements of: connection-pool-params**

XML Tag	Type	Description
<code>statement-timeout</code>	int	The time after which a statement currently being executed will time out. <code>statement-timeout</code> relies on underlying JDBC driver support. The server passes the time specified to the JDBC driver using the <code>java.sql.Statement.setQueryTimeout</code> 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.
<code>profile-harvest-frequency-seconds</code>	int	The number of seconds between diagnostic profile harvest operations. Default: 300.
<code>inactive-connection-timeout-seconds</code>	int	The number of inactive seconds on a reserved connection before the connection is reclaimed and released back into the connection pool. Default: 0.
<code>shrink-frequency-seconds</code>	int	The number of seconds to wait before shrinking a connection pool that has incrementally increased to meet demand. Default: 900.
<code>driver-interceptor</code>	string	Specifies the absolute name of the application class used to intercept method calls to the JDBC driver. The application specified must implement the <code>weblogic.jdbc.extensions.DriverInterceptor</code> interface.
<code>seconds-to-trust-an-idle-connection</code>	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.
<code>pinned-to-thread</code>	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: <code>false</code> .
<code>test-connections-on-reserve</code>	boolean	Test a connection before giving it to a client. Requires that you specify <code>test-table-name</code> . Default: <code>false</code> .

**Table F-5 (Cont.) Child Elements of: connection-pool-params**

XML Tag	Type	Description
profile-type	int	Specifies that type of profile data to be collected.
statement-cache-type	string	The algorithm used for maintaining the prepared statements stored in the statement cache. Valid values: <ul style="list-style-type: none"> <li>■ LRU - when a new prepared or callable statement is used, the least recently used statement is replaced in the cache</li> <li>■ FIXED - the first fixed number of prepared and callable statements are cached</li> </ul> Default: LRU.
connection-reserve-timeout-seconds	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.
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 <code>test-frequency-seconds</code> and <code>enable test-reserved-connections</code> . The default SQL code used to test a connection is <code>select count(*) from test-table-name</code> where <code>test-table-name</code> is the value of the <code>test-table-name</code> element. Most database servers optimize this SQL to avoid a table scan, but it is still a good idea to set <code>test-table-name</code> to the name of a table that is known to have few rows, or even no rows. If <code>test-table-name</code> begins with <code>SQL</code> , 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 <code>SQL</code> 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.

**Table F-5 (Cont.) Child Elements of: connection-pool-params**

XML Tag	Type	Description
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.
highest-number-waiters	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.

## Attributes

The connection-pool-params server configuration element has no attributes.

## Example

The following example shows how to use the connection-pool-params element in the Oracle Event Processing 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>
```

## cql

Use this element to configure Oracle CQL-specific options in the Oracle Event Processing server.

### Child Elements

The `cql` server configuration element supports the following child elements:

- `name`
- `scheduler`
- `partition-order-capacity`

### Attributes

The `cql` server configuration element has no attributes.

### Example

The following example shows how to use the `cql` element in the Oracle Event Processing 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`.

## data-source

This configuration type defines configuration for a `DataSource` service.

### Child Elements

The `data-source` server configuration element supports the following child elements:

- `name`
- `xa-params`
- `data-source-params`
- `connection-pool-params`
- `driver-params`

### Attributes

The `data-source` server configuration element has no attributes.

## Example

The following example shows how to use the `data-source` element in the Oracle Event Processing 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`.

## data-source-params

Use this element to specify data source-related [data-source](#) parameters.

## Child Elements

The `data-source-params` server configuration element supports the child elements that [Table F-6](#) lists.

**Table F-6** Child Elements of: *data-source-params*

XML Tag	Type	Description
<code>algorithm-type</code>	See Description	The algorithm determines the connection request processing for the multi data source. Valid values: <ul style="list-style-type: none"> <li>▪ Failover</li> <li>▪ Load-Balancing</li> </ul> Default: Failover.
<code>stream-chunk-size</code>	int	Specifies the data chunk size for steaming data types between 1 and 65536. Default: 256.
<code>row-prefetch</code>	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.



**Table F-6 (Cont.) Child Elements of: data-source-params**

XML Tag	Type	Description
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.
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: <ul style="list-style-type: none"> <li>element: contains the string name of a valid data-source element. For more information, see <a href="#">data-source</a>.</li> <li>config-data-source-DataSourceParams-JNDINames.</li> </ul>
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 <code>weblogic.jdbc.extensions.ConnectionPoolFailoverCallback</code> interface.
global-transactions-protocol	int	Determines the transaction protocol (global transaction processing behavior) for the data source. Valid values: <ul style="list-style-type: none"> <li>TwoPhaseCommit - Standard XA transaction processing. Requires an XA driver</li> <li>LoggingLastResource - A performance enhancement for one non-XA resource</li> <li>EmulateTwoPhaseCommit - Enables one non-XA resource to participate in a global transaction, but has some risk to data</li> <li>OnePhaseCommit - One-phase XA transaction processing using a non-XA driver. This is the default setting</li> <li>None - Support for local transactions only</li> </ul> Default: OnePhaseCommit.

## Attributes

The data-source-params server configuration element has no attributes.

## Example

The following example shows how to use the data-source-params element in the Oracle Event Processing 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>

```

## driver-params

Use this element to specify JDBC driver-related [data-source](#) parameters.

### Child Elements

The `driver-params` server configuration element supports the child elements that [Table F-7](#) lists.

**Table F-7 Child Elements of: driver-params**

XML Tag	Type	Description
<code>use-xa-data-source-interface</code>	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.
<code>password</code>	string	The password attribute passed to the JDBC driver when creating physical database connections.
<code>driver-name</code>	string	The full package name of JDBC driver class used to create the physical database connections in the connection pool in the data source.
<code>url</code>	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.
<code>properties</code>	string	Specifies the list of properties passed to the JDBC driver when creating physical database connections. This element contains one or more <code>element</code> child elements that contain child elements: <ul style="list-style-type: none"> <li>■ name: the property name.</li> <li>■ value: the property value.</li> </ul>

### Attributes

The `driver-params` server configuration element has no attributes.

## Example

The following example shows how to use the `driver-params` element in the Oracle Event Processing 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>
```

## domain

Use this element to configure a domain name in the Oracle Event Processing server.

### Child Elements

The domain server configuration element supports the following child elements:

- [name](#)

### Attributes

The domain server configuration element has no attributes.

## Example

The following example shows how to use the `domain` element in the Oracle Event Processing server configuration file:

```
<domain>
  <name>WLEventServerDomain</name>
</domain>
```

In the example, the domain's unique identifier is `WLEventServerDomain`.

## debug

Use this element to configure one or more debug properties for the Oracle Event Processing server.

### Child Elements

The debug server configuration element supports the child elements that [Table F–8](#) lists.

**Table F–8 Child Elements of: debug**

XML Tag	Type	Description
name	string	The name of this debug configuration. For more information, see <a href="#">name</a> .
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 Event Processing Debugging Options Using a Configuration File" in the <i>Oracle Fusion Middleware Administrator's Guide for Oracle Event Processing</i>

### Attributes

The debug server configuration element has no attributes.

### 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 Event Processing server configuration file.

```
<debug>
  <name>myDebug</name>
  <debug-properties>
    <DebugSDS>true</DebugSDS>
  ...
  </debug-properties>
</debug>
```

## event-store

Use this element to configure an event store for the Oracle Event Processing server.

### Child Elements

The event-store server configuration element supports the child elements that [Table F–9](#) lists.

**Table F–9 Child Elements of: event-store**

XML Tag	Type	Description
name	string	The name of this debug configuration. For more information, see <a href="#">name</a> .

**Table F–9 (Cont.) Child Elements of: event-store**

XML Tag	Type	Description
provider-order	string	Specifies the name of one or more <code>provider</code> child elements in the order in which the Oracle Event Processing server should access them.  For more information, see: <ul style="list-style-type: none"> <li>▪ <a href="#">rdbms-event-store-provider</a></li> </ul>

## Attributes

The `event-store` server configuration element has no attributes.

## Example

The following example shows how to use the `event-store` element in the Oracle Event Processing 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`.

## 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.

## Child Elements

The `exported-jndi-context` server configuration element supports the child elements that [Table F–10](#) lists.

**Table F–10 Child Elements of: exported-jndi-context**

XML Tag	Type	Description
name	string	The name of this debug configuration. For more information, see <a href="#">name</a> .
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 <a href="#">rmi</a> .

## Attributes

The `exported-jndi-context` server configuration element has no attributes.

## Example

The following example shows how to use the `exported-jndi-context` element in the Oracle Event Processing 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`.

## http-pubsub

Use this element to configure an HTTP publish-subscribe service.

### Child Elements

The `http-pubsub` server configuration element supports the following child elements:

- `name`
- `path`
- `pubsub-bean`

### Attributes

The `http-pubsub` server configuration element has no attributes.

## Example

The following example shows how to use the `http-pubsub` element in the Oracle Event Processing 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>
  </channels>
</http-pubsub>
```

```

        <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`.

## jetty

Use this element to configure an instance of the Jetty HTTP server.

### Child Elements

The `jetty` server configuration element supports the child elements that [Table F-11](#) lists.

**Table F-11** Child Elements of: `jetty`

XML Tag	Type	Description
<code>name</code>	string	The name of this <code>jetty</code> element. For more information, see <a href="#">name</a> .
<code>network-io-name</code>	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.
<code>work-manager-name</code>	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 <a href="#">work-manager</a> .
<code>scratch-directory</code>	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 <code>scratch-directory</code> parameter on the <code>jetty-web-app</code> element. If this directory does not exist, it will be created.
<code>debug-enabled</code>	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.
<code>listen-port</code>	int	The name of the network port that should be set. This parameter may not be set if the <code>network-io-name</code> parameter is not specified. When this parameter is used instead of <code>network-io-name</code> , a simplified socket I/O subsystem is used that does not require the <code>netio</code> module.
<code>secure-network-io-name</code>	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 <code>netio</code> configuration object.

### Attributes

The `jetty` server configuration element has no attributes.

### Example

The following example shows how to use the `jetty` element in the Oracle Event Processing 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`.

## 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.

### Child Elements

The `jetty-web-app` server configuration element supports the child elements that [Table F-12](#) lists.

**Table F-12** Child Elements of: `jetty-web-app`

XML Tag	Type	Description
<code>name</code>	string	The name of this <code>jetty-web-app</code> element. For more information, see <a href="#">name</a> .
<code>context-path</code>	string	The context path where this web app will be deployed in the web server's name space. Default: /
<code>scratch-directory</code>	string	The location where Jetty should store temporary files for this web app. This parameter overrides the <code>scratch-directory</code> parameter on the <code>jetty</code> element. If this directory does not exist, it will be created.
<code>path</code>	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.
<code>jetty-name</code>	string	The name of the Jetty service where this Web application should be deployed. This name must match the name of an existing <code>jetty</code> configuration object. For more information, see <a href="#">jetty</a> .

### Attributes

The `jetty-web-app` server configuration element has no attributes.

### Example

The following example shows how to use the `jetty-web-app` element in the Oracle Event Processing 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`.

## jmx

Use this element to configure Java Management Extension (JMX) properties in the Oracle Event Processing server.



## Child Elements

The `jmx` server configuration element supports the child elements that [Table F–13](#) lists.

**Table F–13** Child Elements of: `jmx`

XML Tag	Type	Description
<code>name</code>	string	The name of this debug configuration. For more information, see <a href="#">name</a> .
<code>rmi-service-name</code>	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 <a href="#">rmi</a> .
<code>jndi-service-name</code>	string	The name of the JNDI service to which the JMX server will bind its object.

## Attributes

The `jmx` server configuration element has no attributes.

## Example

The following example shows how to use the `jmx` element in the Oracle Event Processing 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`.

## 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.

## Child Elements

The `jndi-context` server configuration element supports the child elements that [Table F–14](#) lists.

**Table F–14** Child Elements of: `jndi-context`

XML Tag	Type	Description
<code>name</code>	string	The name of this debug configuration. For more information, see <a href="#">name</a> .
<code>default-provider</code>	string	This parameter defaults to <code>true</code> . If it is set to <code>false</code> 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 <code>JNDIContextType</code> bean in the configuration with <code>DefaultProvider</code> set to <code>true</code> . If multiple <code>JNDIContextType</code> objects are placed in the configuration, the effect will be the same as if one was started.

## Attributes

The `jndi-context` server configuration element has no attributes.

## Example

The following example shows how to use the `jndi-context` element in the Oracle Event Processing 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`.

## log-file

Use this element to configure logging to a file on the Oracle Event Processing server.

### Child Elements

The `log-file` server configuration element supports the child elements that [Table F-15](#) lists.

**Table F-15** Child Elements of: `log-file`

XML Tag	Type	Description
<code>name</code>	string	The name of this work-manager element. For more information, see <a href="#">name</a> .
<code>number-of-files-limited</code>	boolean	Determines whether old rotated files need to be kept around forever. Default: <code>false</code> .
<code>rotation-type</code>	string	Determines how the log file rotation will be performed based on size, time or not at all. Valid values: <ul style="list-style-type: none"> <li>▪ <code>bySize</code></li> <li>▪ <code>byTime</code></li> <li>▪ <code>none</code></li> </ul> Default: <code>bySize</code> .
<code>rotation-time</code>	string	The time in <code>k:mm</code> format when the first rotation happens where <code>k</code> is the hour specified in 24-hour notation and <code>mm</code> is the minutes. Default: <code>00:00</code> .
<code>rotated-file-count</code>	int	If old rotated files are to be deleted, this parameter determines how many of the last files to always keep. Default: <code>7</code> .
<code>rotation-size</code>	int	The size threshold at which the log file is rotated in KB. Default: <code>500</code> .
<code>rotation-time-span-factor</code>	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: <code>(long)(3600 * 1000)</code> .
<code>rotation-time-span</code>	int	The interval for every time based log rotation. Default: <code>24</code> .
<code>base-log-file-name</code>	string	Log file name. Default: <code>server.log</code>
<code>rotate-log-on-startup-enabled</code>	boolean	Specifies whether the log file will be rotated on startup. Default: <code>true</code> .

**Table F–15 (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 <code>Info</code> so that to see <code>Debug</code> and <code>Trace</code> messages you need to ensure that the severity is set to either <code>Debug</code> or <code>Trace</code> . Valid values: <ul style="list-style-type: none"> <li>▪ Emergency</li> <li>▪ Alert</li> <li>▪ Critical</li> <li>▪ Error</li> <li>▪ Warning</li> <li>▪ Notice</li> <li>▪ Info</li> <li>▪ Debug</li> <li>▪ Trace</li> </ul> Default: <code>Notice</code> .
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.

## Attributes

The `log-file` server configuration element has no attributes.

## Example

The following example shows how to use the `log-file` element in the Oracle Event Processing 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`.

## log-stdout

Use this element to configure logging to standard out (console) on the Oracle Event Processing server.

## Child Elements

The `log-stdout` server configuration element supports the child elements that [Table F–16](#) lists.

**Table F–16 Child Elements of: log-stdout**

XML Tag	Type	Description
name	string	The name of this work-manager element. For more information, see <a href="#">name</a> .
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 <code>-1</code> means all frames are displayed. Default: <code>-1</code> .

**Table F–16 (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 <code>Info</code> so that to see <code>Debug</code> and <code>Trace</code> messages you need to ensure that the severity is set to either <code>Debug</code> or <code>Trace</code> . Valid values: <ul style="list-style-type: none"> <li>▪ Emergency</li> <li>▪ Alert</li> <li>▪ Critical</li> <li>▪ Error</li> <li>▪ Warning</li> <li>▪ Notice</li> <li>▪ Info</li> <li>▪ Debug</li> <li>▪ Trace</li> </ul> Default: <code>Notice</code> .

## Attributes

The `log-stdout` server configuration element has no attributes.

## Example

The following example shows how to use the `log-stdout` element in the Oracle Event Processing 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`.

## logging-service

Use this element to configure a logging service on the Oracle Event Processing server.

## Child Elements

The `logging-service` server configuration element supports the child elements that [Table F–17](#) lists.

**Table F–17 Child Elements of: logging-service**

XML Tag	Type	Description
name	string	The name of this <code>work-manager</code> element. For more information, see <a href="#">name</a> .
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-17 (Cont.) Child Elements of: logging-service**

XML Tag	Type	Description
logger-severity	string	<p>Defines the threshold importance of the messages that are propagated to the handlers. The default is <code>Info</code> so that to see <code>Debug</code> and <code>Trace</code> messages you need to ensure that the severity is set to either <code>Debug</code> or <code>Trace</code>. Valid values:</p> <ul style="list-style-type: none"> <li>▪ <code>Emergency</code></li> <li>▪ <code>Alert</code></li> <li>▪ <code>Critical</code></li> <li>▪ <code>Error</code></li> <li>▪ <code>Warning</code></li> <li>▪ <code>Notice</code></li> <li>▪ <code>Info</code></li> <li>▪ <code>Debug</code></li> <li>▪ <code>Trace</code></li> </ul> <p>Default: <code>Info</code>.</p>
logger-severity-properties	See Description	The Severity values for different nodes in the <code>Logger</code> tree composed of one or more <code>entry</code> child elements each containing a <code>key</code> and <code>value</code> child element.

## Attributes

The `logging-service` server configuration element has no attributes.

## Example

The following example shows how to use the `logging-service` element in the Oracle Event Processing 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`.

## message-filters

Use this element to configure one or more message filters for a [pubsub-bean](#) element.

## 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.

## Attributes

The `message-filters` server configuration element has no attributes.

## Example

The following example shows how to use the `message-filters` element in the Oracle Event Processing server configuration file:

```
<http-pubsub>
  <name>pubsub</name>
  <path>/pubsub</path>
  <pub-sub-bean>
  ...
    <message-filters>
      <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>
```

## name

Use this element to declare a unique identifier for an Oracle Event Processing server configuration element.

## Child Elements

The `name` server configuration element has no child elements.

## Attributes

The `name` server configuration element has no attributes.

## Example

The following example shows how to use the `name` element in the Oracle Event Processing server configuration file:

```
<http-pubsub>
  <name>pubsub</name>
  <path>/pubsub</path>
  ...
</http-pubsub>
```

## netio

Use this element to represent a network input/output (IO) service, that may be used by other services to act as the server for network IO.

## Child Elements

The netio server configuration element supports the child elements that [Table F-18](#) lists.

**Table F-18 Child Elements of: netio**

XML Tag	Type	Description
name	string	The name of this netio element. For more information, see <a href="#">name</a> .
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. For a list of the valid provider types, see "Network IO Providers" in the <i>Oracle Fusion Middleware Administrator's Guide for Oracle Event Processing</i> .
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.

## Attributes

The netio server configuration element has no attributes.

## Example

The following example shows how to use the netio element in the Oracle Event Processing server configuration file:

```
<netio>
  <name>myNetio</name>
  <port>12345</port>
</netio>
```

In the example, the netio element's unique identifier is myNetio.

## netio-client

Use this element to register a network input/output (IO) service that may be used to perform non-blocking network IO, but which will not act as a server and listen for incoming connections.

## Child Elements

The netio-client server configuration element supports the child elements that [Table F-19](#) lists.

**Table F-19 Child Elements of: netio-client**

XML Tag	Type	Description
name	string	The name of this netio element. For more information, see <a href="#">name</a> .

**Table F–19 (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. For a list of the valid provider types, see "Network IO Providers" in the <i>Oracle Fusion Middleware Administrator's Guide for Oracle Event Processing</i> .

## Attributes

The netio-client server configuration element has no attributes.

## Example

The following example shows how to use the netio-client element in the Oracle Event Processing 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.

## partition-order-capacity

Use this element to define the maximum capacity of a query partition when the ordering-constraint attribute is set to PARTITION\_ORDERED. Set this on a [cql](#) component. Consider setting this element's value when you've configured a query processor for parallel execution, and when the query's ordering-constraint attribute is set to PARTITION\_ORDERED.

The element's default value is 4.

For more information, including best practices and information on the locations where this value can be set (including their precedence), see "Using partition-order-capacity with Partitioning Queries" in [Chapter 17, "Querying an Event Stream with Oracle CQL"](#).

## Child Elements

The partition-order-capacity element has no child elements.

## Attributes

The partition-order-capacity element has no attributes.

## Example

The following example shows how to use the partition-order-capacity element in the Oracle Event Processing server configuration file:

```
<cql>
  <name>myCQL</name>
  <partition-order-capacity>20</partition-order-capacity>
</cql>
```



## path

Use this element to configure the path for an [http-pubsub](#) element.

### Child Elements

The path element has no child elements.

### Attributes

The path element has no attributes.

### Example

The following example shows how to use the path element in the Oracle Event Processing 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>
```

## pubsub-bean

Use this element to configure a publish-subscribe bean for an [http-pubsub](#) element.

### Child Elements

The pubsub-bean server configuration element supports the following child elements:

- [name](#)
- [server-config](#)
- [message-filters](#)
- [channels](#)

- [channel-constraints](#)
- [services](#)

## Attributes

The pubsub-bean server configuration element has no attributes.

## Example

The following example shows how to use the pubsub-bean element in the Oracle Event Processing 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>
```

## rdbms-event-store-provider

Use this element to configure an event store provider that uses a relational database management system in the Oracle Event Processing server.

By default, Oracle Event Processing server uses a Berkeley database instance as the event store provider as [Section , "bdb-config"](#) describes.

## Child Elements

The rdbms-event-store-provider server configuration element supports the child elements that [Table F-20](#) lists.

**Table F-20** Child Elements of: rdbms-event-store-provider

XML Tag	Type	Description
name	string	The name of this debug configuration. For more information, see <a href="#">name</a> .

**Table F–20 (Cont.) Child Elements of: rdbms-event-store-provider**

XML Tag	Type	Description
init-timeout	int	The maximum time (in milliseconds) that the Oracle Event Processing 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 <a href="#">data-source</a> .
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.

## Attributes

The rdbms-event-store-provider server configuration element has no attributes.

## Example

The following example shows how to use the rdbms-event-store-provider element in the Oracle Event Processing 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>
  </user-policy-attributes>
</rdbms-event-store-provider>
```

In the example, the rdbms-event-store-provider element's unique identifier is test-rdbms-provider.

## rmi

Use this element to configure an RMI service, which allows server-side objects to be exported to remote clients.

## Child Elements

The rmi server configuration element supports the child elements that [Table F–21](#) lists.

**Table F–21 Child Elements of: rmi**

XML Tag	Type	Description
name	string	The name of this rmi element. For more information, see <a href="#">name</a> .
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.

**Table F–21 (Cont.) Child Elements of: rmi**

XML Tag	Type	Description
heartbeat-interval	int	The time in milliseconds between heartbeats. Once the number of unsuccessful heartbeat attempts has reached the value specified by the <code>HeartbeatPeriod</code> attribute, all registered <code>DisconnectListener</code> instances will be notified. Default-Value: 5000.

## Attributes

The `rmi` server configuration element has no attributes.

## Example

The following example shows how to use the `rmi` element in the Oracle Event Processing 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`.

## scheduler

Use this element to configure `cql` scheduler options in the Oracle Event Processing server.

## Child Elements

The scheduler server configuration element supports the child elements that [Table F–22](#) lists.

**Table F–22 Child Elements of: scheduler**

XML Tag	Type	Description
class-name	string	Specify the value for one of the <code>sched_name</code> scheduling option as the fully qualified package name of Java class that implements the Oracle Event Processing Service Engine scheduling algorithm. This class determines in what order the Oracle Event Processing Service Engine scheduler executes Oracle CQL queries. Valid values: <ul style="list-style-type: none"> <li>▪ <code>oracle.cep.execution.scheduler.RoundRobinScheduler</code>: 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.</li> <li>▪ <code>oracle.cep.execution.scheduler.FIFOScheduler</code>: 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.</li> </ul> Default: <code>oracle.cep.execution.scheduler.RoundRobinScheduler</code>

**Table F–22 (Cont.) Child Elements of: scheduler**

XML Tag	Type	Description
runtime	long	Total number of seconds that the Oracle Event Processing Service Engine scheduler will run. Default: 1000000 ms.
time-slice	int	The frequency at which the Oracle Event Processing Service Engine scheduler executes Oracle CQL queries. Default: 1000 ms
schedule-on-new-thread	boolean	Whether or not the Oracle Event Processing Service Engine scheduler will use a separate thread. Options are: <ul style="list-style-type: none"> <li>▪ true: the scheduler runs in a separate thread.</li> <li>▪ false: the scheduler runs in the same thread as the <b>Oracle Event Processing Service Engine</b> (Default).</li> </ul>

## Attributes

The scheduler server configuration element has no attributes.

## Example

The following example shows how to use the `scheduler` element in the Oracle Event Processing server configuration file:

```
<cql>
  <name>myCQL</name>
  <scheduler>
    <class-name>oracle.cep.execution.scheduler.FIFOScheduler</class-name>
  </scheduler>
</cql>
```

## server-config

Use this element to configure the server-specific properties of a [pubsub-bean](#) element.

## Child Elements

The `server-config` server configuration element supports the child elements that [Table F–23](#) lists.

**Table F–23 Child Elements of: server-config**

XML Tag	Type	Description
name	string	The name of this <code>server-config</code> element. For more information, see <a href="#">name</a> .

**Table F–23 (Cont.) Child Elements of: server-config**

XML Tag	Type	Description
supported-transport	See Description	<p>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:</p> <ul style="list-style-type: none"> <li>▪ long-polling: Using this transport, the client requests information from Oracle Event Processing server and if Oracle Event Processing server does not have information available, it does not reply until it has. When the Oracle Event Processing server replies, the client typically sends another request immediately.</li> <li>▪ 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 Bayeux protocol. For more information on the Bayeux protocol, see <a href="http://svn.cometd.org/trunk/bayeux/bayeux.html">http://svn.cometd.org/trunk/bayeux/bayeux.html</a>.</li> </ul> <p>For more information, see "How the HTTP Pub-Sub Server Works" in the <i>Oracle Fusion Middleware Administrator's Guide for Oracle Event Processing</i>.</p>
client-timeout-secs	int	<p>Specifies the number of seconds after which the HTTP pub-sub server disconnects a client if the client does not send back a connect/reconnect message.</p> <p>Default: 60.</p>
persistent-client-timeout-secs	int	<p>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.</p> <p>Default: 600 seconds.</p>
interval-millisecs	int	<p>Specifies how long (in milliseconds) the client can delay subsequent requests to the <code>/meta/connect</code> channel.</p> <p>Default: 500 ms.</p>
work-manager	string	<p>Specifies the name of the work manager that delivers messages to clients. The value of this element corresponds to the value of the <code>name</code> child element of the <code>work-manager</code> you want to assign.</p> <p>For more information, see <a href="#">work-manager</a>.</p>
publish-without-connect-allowed	boolean	<p>Specifies whether clients can publish messages without having explicitly connected to the HTTP pub-sub server. Valid values:</p> <ul style="list-style-type: none"> <li>▪ true</li> <li>▪ false</li> </ul>

## Attributes

The `server-config` server configuration element has no attributes.

## Example

The following example shows how to use the `server-config` element in the Oracle Event Processing 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>

```

## services

Use this element to configure the service properties of a [pubsub-bean](#) element.

### Child Elements

The `services` server configuration element contains one or more `element` child elements that each support the child elements that [Table F-24](#) lists.

**Table F-24** Child Elements of: `services`

XML Tag	Type	Description
<code>service-channel</code>	string	Specifies a service channel, for example: <code>/service/echo</code> .
<code>service-class</code>	string	Specifies the class to service this service, for example: <code>EchoService</code> .
<code>service-method</code>	string	Define a service method in the service class. The service method must have only one payload parameter of type <code>Object</code> . For example: <code>Object echo(Object payload)</code> .

### Attributes

The `services` server configuration element has no attributes.

### Example

The following example shows how to use the `services` element in the Oracle Event Processing 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>

```

## show-detail-error-message

Use this element to configure whether or not the Oracle Event Processing server uses secure connections.

### Child Elements

The `show-detail-error-message` server configuration element supports the child elements that [Table F-25](#) lists.

**Table F-25** Child Elements of: `show-detail-error-message`

XML Tag	Type	Description
<code>name</code>	string	The name of this <code>show-detail-error-message</code> element. For more information, see <a href="#">name</a> .
<code>value</code>	boolean	Whether or not to show detailed error messages. Valid values: <ul style="list-style-type: none"> <li>▪ <code>true</code>: the Oracle Event Processing server shows detailed error messages.</li> <li>▪ <code>false</code>: the Oracle Event Processing server shows abbreviated error messages (default).</li> </ul>

### Attributes

The `show-detail-error-message` server configuration element has no attributes.

### Example

The following example shows how to use the `show-detail-error-message` element in the Oracle Event Processing 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`.

## ssl

Use this element to configure Secure Sockets Layer-specific properties on the Oracle Event Processing server.

### Child Elements

The `ssl` server configuration element supports the child elements that [Table F-26](#) lists.

**Table F-26** Child Elements of: *ssl*

XML Tag	Type	Description
<code>name</code>	string	The name of this cluster. For more information, see <a href="#">name</a> .
<code>key-store</code>	string	Specifies the file path to the key store such as <code>./ssl/evsidentity.jks</code> .
<code>key-store-pass</code>	See Description	This element contains a <code>password</code> child element with a string value that specifies the password used to access the key store.
<code>key-store-alias</code>	string	Specifies the alias for the key store.
<code>key-manager-algorithm</code>	string	Specifies the key manager algorithm such as <code>SunX509</code> .
<code>ssl-protocol</code>	string	Specifies the SSL protocol such as <code>TLS</code> .
<code>trust-store</code>	string	Specifies the file path to the trust store such as <code>./ssl/evstrust.jks</code> .
<code>trust-store-pass</code>	See Description	This element contains a <code>password</code> child element with a string value that specifies the password used to access the trust store.
<code>trust-store-alias</code>	string	Specifies the alias for the trust store.
<code>trust-store-type</code>	string	Specifies the trust store type such as <code>JKS</code> .
<code>trust-manager-algorithm</code>	string	Specifies the trust manager algorithm such as <code>SunX509</code> .
<code>enforce-fips</code>	boolean	Specifies whether or not Oracle Event Processing server uses a Federal Information Processing Standards (FIPS)-certified pseudo-random number generator.  For more information, see "FIPS" in the <i>Oracle Fusion Middleware Administrator's Guide for Oracle Event Processing</i> .
<code>need-client-auth</code>	boolean	Specifies whether or not client certificate authentication is required.
<code>ciphers</code>	See Description	This element contains one or more <code>cipher</code> child elements, each with a string value that specifies the ciphers that are required.
<code>secure-random-algorithm</code>	string	When <code>enforce-fips</code> is set to <code>true</code> , specify the secure random algorithm to use. Valid values: <ul style="list-style-type: none"> <li>▪ <code>FIPS186PRNG</code></li> </ul>
<code>secure-random-provider</code>	string	When <code>enforce-fips</code> is set to <code>true</code> , specify the secure random provider to use. Valid values: <ul style="list-style-type: none"> <li>▪ <code>JsafeJCE</code></li> </ul>

## Attributes

The `ssl` server configuration element has no attributes.

## Example

The following example shows how to use the `ssl` element in the Oracle Event Processing 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`.

## timeout-seconds

Use this element to configure [weblogic-jta-gateway](#) default transaction timeout in seconds in the Oracle Event Processing server.

Default: 60.

## Child Elements

The `timeout-seconds` server configuration element has no attributes.

## Attributes

The `timeout-seconds` server configuration element has no attributes.

## Example

The following example shows how to use the `timeout-seconds` element in the Oracle Event Processing 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>
```

## transaction-manager

Use this element to configure transaction manager properties in the Oracle Event Processing server.

### Child Elements

The transaction-manager server configuration element supports the child elements that [Table F-27](#) lists.

**Table F-27 Child Elements of: transaction-manager**

XML Tag	Type	Description
name	string	The name of this transaction-manager element. For more information, see <a href="#">name</a> .
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: <ul style="list-style-type: none"> <li>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.</li> <li>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.</li> <li>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.</li> </ul> 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–27 (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 <a href="#">rmi</a> .
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–27 (Cont.) Child Elements of: transaction-manager**

XML Tag	Type	Description
recovery-threshold-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-dispatch-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.

## Attributes

The transaction-manager server configuration element has no attributes.

## Example

The following example shows how to use the transaction-manager element in the Oracle Event Processing 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_cel</rmi-service-name>
</transaction-manager>
```

In the example, the transaction-manager element's unique identifier is My\_tm.

## use-secure-connections

Use this element to configure whether or not the Oracle Event Processing server uses secure connections.

For more information, see "How to Configure SSL in a Multi-Server Domain for Oracle Event Processing Visualizer" in the *Oracle Fusion Middleware Administrator's Guide for Oracle Event Processing*.

## Child Elements

The `use-secure-connections` server configuration element supports the child elements that [Table F-28](#) lists.

**Table F-28 Child Elements of: `use-secure-connections`**

XML Tag	Type	Description
<code>name</code>	string	The name of this <code>use-secure-connections</code> element. For more information, see <a href="#">name</a> .
<code>value</code>	boolean	Whether or not to use secure connections. Valid values: <ul style="list-style-type: none"> <li>▪ <code>true</code>: the Oracle Event Processing server uses only secure connections.</li> <li>▪ <code>false</code>: the Oracle Event Processing server accepts connections that are not secure.</li> </ul>

## Attributes

The `use-secure-connections` server configuration element has no attributes.

## Example

The following example shows how to use the `use-secure-connections` element in the Oracle Event Processing 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`.

## weblogic-instances

Use this element to configure Oracle Event Processing server instances for a [weblogic-jta-gateway](#) element.

## 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
<code>domain-name</code>	string	Specifies the name of the domain of the Oracle Event Processing server.
<code>server-name</code>	string	Specifies the name of the Oracle Event Processing server.
<code>protocol</code>	string	Specifies the JTA protocol. Default: <code>t3</code> .
<code>host-address</code>	string	The host name or IP address of the Oracle Event Processing server.
<code>port</code>	int	The <a href="#">netio</a> port for the Oracle Event Processing server.

## Attributes

The `weblogic-instances` server configuration element has no attributes.

## Example

The following example shows how to use the `weblogic-instances` element in the Oracle Event Processing 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>
```

## weblogic-jta-gateway

Use this element to configure the attributes for the singleton Oracle Event Processing server client JTA gateway service.

## Child Elements

The `weblogic-jta-gateway` server configuration element supports the following child elements:

- `name`
- `timeout-seconds`
- `weblogic-instances`

## Attributes

The `weblogic-jta-gateway` server configuration element has no attributes.

## Example

The following example shows how to use the `weblogic-jta-gateway` element in the Oracle Event Processing 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`.

## weblogic-rmi-client

Use this element to configure the attributes for the singleton Oracle Event Processing server RMI client.

### 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
<code>name</code>	string	The name of this <code>weblogic-rmi-client</code> element. For more information, see <a href="#">name</a> .
<code>netio-name</code>	string	Specifies the name of the <code>netio-client</code> element to use. For more information, see <a href="#">netio-client</a> .
<code>secure-netio-name</code>	string	Specifies the name of the <code>netio-client</code> element configured for SSL. For more information, see <a href="#">netio-client</a> .

### Attributes

The `weblogic-rmi-client` server configuration element has no attributes.

### Example

The following example shows how to use the `weblogic-rmi-client` element in the Oracle Event Processing 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`.

## work-manager

Use this element to configure a work manager on the Oracle Event Processing server.

### 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 <a href="#">name</a> .
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.

## Attributes

The work-manager server configuration element has no attributes.

## Example

The following example shows how to use the work-manager element in the Oracle Event Processing 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.

## xa-params

Use this element to specify distributed transaction-related [data-source](#) parameters.

## 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.

**Table F–32 (Cont.) Child Elements of: xa-params**

XML Tag	Type	Description
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 <code>XAResource.setTransactionTimeout</code> 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 <code>xa-set-transaction-timeout</code> to enable setting the transaction branch timeout. Default: 0.
rollback-local-tx-upon-conn-close	boolean	Enables the server to call <code>rollback</code> 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: <code>false</code> .
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 <code>xa-transaction-timeout</code> . When enabled, the Transaction Manager calls <code>XAResource.setTransactionTimeout</code> before calling <code>XAResource.start</code> , 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: <code>false</code> .
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: <code>false</code> .
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 <code>MaxXACallMillis</code> , 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: <code>true</code> .
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: <code>false</code> .
xa-end-only-once	boolean	Specifies that <code>XAResource.end</code> is called only once for each pending <code>XAResource.start</code> . This option prevents the XA driver from calling <code>XAResource.end(TMSUSPEND)</code> and <code>XAResource.end(TMSUCCESS)</code> successively. Only applies to data sources that use an XA driver. Use this setting to work around specific problems with JDBC XA drivers. Default: <code>false</code> .
xa-retry-interval-seconds	int	The number of seconds between XA retry operations if <code>XARetryDurationSeconds</code> is set to a positive value. Default: 60.

**Table F-32 (Cont.) Child Elements of: xa-params**

XML Tag	Type	Description
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: <i>false</i> .
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: <i>false</i> .

## Attributes

The xa-params server configuration element has no attributes.

## Example

The following example shows how to use the xa-params element in the Oracle Event Processing 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>
```



---

## Schema Reference: Message Catalog msgcat.dtd

This appendix provides a reference to elements of the `msgcat.dtd` schema, the schema behind XML you use to configure message catalog elements for log messages you can localize.

This appendix includes the following sections:

- [Overview of the Message Catalog Elements](#)
- [message\\_catalog](#)
- [logmessage](#)
- [message](#)
- [messagebody](#)
- [messagedetail](#)
- [cause](#)
- [action](#)

### Overview of the Message Catalog Elements

Oracle Event Processing provides a number of message catalog elements that you use to define log messages that you can localize.

### Element Hierarchy

The top-level Oracle Event Processing message catalog elements are organized into the following hierarchies, depending on message catalog type:

- [Example G–1, "Log Message Catalog Hierarchy"](#)
- [Example G–2, "Simple Text Catalog Hierarchy"](#)

#### ***Example G–1 Log Message Catalog Hierarchy***

```
message_catalog
  log_message
    messagebody
    messagedetail
    cause
    action
```

**Example G–2 Simple Text Catalog Hierarchy**

```
message_catalog
  message
    messagebody
```

## Examples

This section provides the following message catalog examples:

- [Example G–3, "Log Message Catalog"](#)
- [Example G–4, "Simple Text Catalog"](#)

**Example G–3 Log Message Catalog**

```
<?xml version="1.0"?>
<!DOCTYPE message_catalog PUBLIC "weblogic-message-catalog-dtd"
"http://www.bea.com/servers/wls90/dtd/msgcat.dtd">
<message_catalog
  l10n_package="programs.utils"
  i18n_package="programs.utils"
  subsystem="MYUTIL"
  version="1.0"
  baseid="600000"
  endid="600100">
  <logmessage
    messageid="600001"
    severity="warning"
    method="logNoAuthorization(String arg0, java.util.Date arg1,int arg2)">
    <messagebody>
      Could not open file, {0} on {1,date} after {2,number} attempts.
    </messagebody>
    <messagedetail>
      The configuration for this application will be defaulted to
      factory settings. Custom configuration information resides
      in file, {0}, created on {1,date}, but is not readable.
    </messagedetail>
    <cause>
      The user is not authorized to use custom configurations. Custom
      configuration information resides in file, {0}, created on
      {1,date}, but is not readable.The attempt has been logged to
      the security log.
    </cause>
    <action>
      The user needs to gain appropriate authorization or learn to
      live with the default settings.
    </action>
  </logmessage>
</message_catalog>
```

**Example G–4 Simple Text Catalog**

```
<?xml version="1.0"?>
<!DOCTYPE message_catalog PUBLIC "weblogic-message-catalog-dtd"
"http://www.bea.com/servers/wls90/dtd/msgcat.dtd">
<message_catalog
  l10n_package="programs.utils"
  i18n_package="programs.utils"
  subsystem="MYUTIL"
  version="1.0">
  <message messageid="FileMenuTitle">
    <messagebody>
      File
    </messagebody>
  </message>
```

&lt;/message\_catalog&gt;

## message\_catalog

This element represents the log message catalog.

### Child Elements

The message\_catalog element supports the following child elements:

- [message](#)
- [logmessage](#)

### Attributes

Table G-1 lists the attributes of the message\_catalog element.

**Table G-1 Attributes of the message\_catalog Element**

Attribute	Description	Data Type	Required?
i18n_package	Java package containing generated Logger classes for this catalog. The classes are named after the catalog file name. For example, for a catalog using mycat.xml, a generated Logger class would be called <code>i18n_package.mycatLogger.class</code> . Syntax: standard Java package syntax. Example: <code>i18n_package="programs.utils"</code> Default: <code>weblogic.i18n</code>	String	No.
l10n_package	A Java package containing generated LogLocalizer properties for the catalog. For example, for a catalog called mycat.xml, the following property files would be generated: <ul style="list-style-type: none"> <li>▪ <code>l10n_package.mycatLogLocalizer.properties</code></li> <li>▪ <code>l10n_package.mycatLogLocalizerDetail.properties</code></li> </ul> Syntax: standard Java package syntax. Example: <code>l10n_package="programs.utils"</code> Default: <code>weblogic.i18n</code>	String	No.
subsystem	An acronym identifying the subsystem associated with this catalog. The name of the subsystem is included in the server log and is used for message isolation purposes. Example: <code>subsystem="MYUTIL"</code>	String	Yes.
version	Specifies the version of the msgcat.dtd being used. Use: Must be "1.0" Syntax: <code>x.y</code> where <code>x</code> and <code>y</code> are numeric. Example: <code>version="1.0"</code>	String	Yes.
baseid	Specifies the lowest message ID used in this catalog. Syntax: one to six decimal digits. Example: <code>baseid="600000"</code> Valid values are: <ul style="list-style-type: none"> <li>▪ 000000 for Oracle Event Processing server catalogs</li> <li>▪ 500000 for user-defined catalogs</li> </ul>	String	No.

**Table G-1 (Cont.) Attributes of the message\_catalog Element**

Attribute	Description	Data Type	Required?
endid	Specifies the highest message ID used in this catalog. Syntax: one to six decimal digits. Example: endid="600100" Valid values are: <ul style="list-style-type: none"> <li>499999 for Oracle Event Processing server catalogs</li> <li>999999 for user-defined catalogs</li> </ul>	String	No.
loggables	Indicates whether to generate additional methods that return loggable objects. Example: loggable="true" Valid values are: <ul style="list-style-type: none"> <li>true</li> <li>false</li> </ul> Default: false	String	
prefix	Specifies a String to be prepended to message IDs when logged. Oracle Event Processing server messages default to "CEP" as the prefix and may not specify a different prefix. User messages can specify any prefix. A prefixed message ID is presented in a log entry as follows: <[prefix-]id>  Where prefix is this attribute and id is the six-digit message ID associated with a specific message. For example, if prefix is "XYZ", then message 987654 would be shown in a log entry as <XYZ-987654>. If prefix is not defined, then the log entry would be <987654>. Syntax: any String (should be limited to five characters). Example: prefix="CEP" Valid values are: <ul style="list-style-type: none"> <li>"CEP" for Oracle Event Processing server catalogs.</li> <li>null for user-defined catalogs</li> </ul>	String	No.
description	An optional attribute that serves to document the catalog content. Example: description="Contains messages logged by the foobar application"	String	No.

## Example

The following example shows how to use the message\_catalog element in log message catalog file:

```
<?xml version="1.0"?>
<!DOCTYPE message_catalog PUBLIC "weblogic-message-catalog-dtd"
"http://www.bea.com/servers/wls90/dtd/msgcat.dtd">
<message_catalog
  l10n_package="programs.utils"
  i18n_package="programs.utils"
  subsystem="MYUTIL"
  version="1.0"
  baseid="600000"
  endid="600100">
  <logmessage
    messageid="600001"
    severity="warning"
    method="logNoAuthorization(String arg0, java.util.Date arg1,int arg2)">
```



```

<messagebody>
  Could not open file, {0} on {1,date} after {2,number} attempts.
</messagebody>
<messagedetail>
  The configuration for this application will be defaulted to
  factory settings. Custom configuration information resides
  in file, {0}, created on {1,date}, but is not readable.
</messagedetail>
<cause>
  The user is not authorized to use custom configurations. Custom
  configuration information resides in file, {0}, created on
  {1,date}, but is not readable.The attempt has been logged to
  the security log.
</cause>
<action>
  The user needs to gain appropriate authorization or learn to
  live with the default settings.
</action>
</logmessage>
</message_catalog>

```

## logmessage

Use this element to define a formal log message.

### Child Elements

The logmessage component configuration element supports the following child elements:

- [messagebody](#)
- [messagedetail](#)
- [cause](#)
- [action](#)

### Attributes

Table G–2 lists the attributes of the logmessage component configuration element.

**Table G–2 Attributes of the logmessage Element**

Attribute	Description	Data Type	Required?
messageid	<p>Unique identifier for this log message. Uniqueness should extend across all catalogs. Value must be in range defined by baseid and endid attributes.</p> <p>Use: Value must be in the range defined by the baseid and endid attributes defined in the message_catalog element.</p> <p>Syntax: one to six decimal digits.</p> <p>Example: messageid="600001"</p>	String	Yes.
datelastchanged	<p>Date/time stamp used for managing modifications to this message. The date is supplied by utilities that run on the catalogs.</p> <p>Use: The date is supplied by utilities that run on the catalogs.</p> <p>Syntax: Long.toString(new Date().getTime());</p>	String	No.

**Table G–2 (Cont.) Attributes of the logmessage Element**

Attribute	Description	Data Type	Required?
severity	<p>Indicates the severity of the log message. User-defined catalogs may only use debug, info, warning, and error.</p> <p>Valid values:</p> <ul style="list-style-type: none"> <li>▪ error</li> <li>▪ warning</li> <li>▪ info</li> <li>▪ debug</li> </ul> <p>Example: severity="warning"</p>	String	Yes.
method	<p>Method signature for logging this message.</p> <p>The syntax is the standard Java method signature, without the qualifiers, semicolon, and extensions. Argument types can be any Java primitive or class. Classes must be fully qualified if not in <code>java.lang</code>. Classes must also conform to <code>java.text.MessageFormat</code> conventions. In general, class arguments should have a useful <code>toString()</code> method.</p> <p>Arguments can be any valid name, but should follow the convention of <code>argn</code> where <code>n</code> is 0 through 9. There can be no more than 10 arguments. For each <code>argn</code> there should be at least one corresponding placeholder in the text elements <a href="#">Section , "Child Elements"</a> describes. Placeholders are of the form <code>{n}</code>, <code>{n,number}</code> or <code>{n,date}</code>.</p>	String	Yes.
methodtype	<p>Specifies type of method to generate. Methods can be:</p> <ul style="list-style-type: none"> <li>▪ Logger methods format the message body into the default locale and log the results.</li> <li>▪ Getter methods return the message body prefixed by the subsystem and messageid, as follows: <code>[subsystem:msgid]text</code>.</li> </ul> <p>Valid values are:</p> <ul style="list-style-type: none"> <li>▪ logger</li> <li>▪ getter</li> </ul> <p>Default: logger.</p>	String	No.
stacktrace	<p>Indicates whether to generate a stack trace for Throwable arguments. When the value is <code>true</code>, a trace is generated.</p> <p>Valid values are:</p> <ul style="list-style-type: none"> <li>▪ true</li> <li>▪ false</li> </ul> <p>Default: true.</p>	String	No.
retired	<p>Indicates whether message is retired. A retired message is one that was used in a previous release but is now obsolete and not used in the current version. Retired messages are not represented in any generated classes or resource bundles.</p> <p>Valid values are:</p> <ul style="list-style-type: none"> <li>▪ true</li> <li>▪ false</li> </ul> <p>Default: false.</p>	String	No.

## Example

The following example shows how to use the `logmessage` element in log message catalog file:

```
<?xml version="1.0"?>
<!DOCTYPE message_catalog PUBLIC "weblogic-message-catalog-dtd"
"http://www.bea.com/servers/wls90/dtd/msgcat.dtd">
<message_catalog
  l10n_package="programs.utils"
  i18n_package="programs.utils"
  subsystem="MYUTIL"
  version="1.0"
  baseid="600000"
  endid="600100">
  <logmessage
    messageid="600001"
    severity="warning"
    method="logNoAuthorization(String arg0, java.util.Date arg1,int arg2)">
    <messagebody>
      Could not open file, {0} on {1,date} after {2,number} attempts.
    </messagebody>
    <messagedetail>
      The configuration for this application will be defaulted to
      factory settings. Custom configuration information resides
      in file, {0}, created on {1,date}, but is not readable.
    </messagedetail>
    <cause>
      The user is not authorized to use custom configurations. Custom
      configuration information resides in file, {0}, created on
      {1,date}, but is not readable.The attempt has been logged to
      the security log.
    </cause>
    <action>
      The user needs to gain appropriate authorization or learn to
      live with the default settings.
    </action>
  </logmessage>
</message_catalog>
```

## message

Use this element to define an informal log message.

### Child Elements

The `message` element supports the following child elements:

- [messagebody](#)

### Attributes

[Table G-3](#) lists the attributes of the `message` element.

**Table G-3 Attributes of the message Element**

Attribute	Description	Data Type	Required?
messageid	Unique identifier for this log message in alpha-numeric string format. Uniqueness is required only within the context of this catalog.	String	Yes.

**Table G-3 (Cont.) Attributes of the message Element**

Attribute	Description	Data Type	Required?
datelastchanged	Date/time stamp used for managing modifications to this message. The date is supplied by utilities that run on the catalogs.  Use: The date is supplied by utilities that run on the catalogs.  Syntax: <code>Long.toString(new Date().getTime());</code>	String	No.
method	Method signature for formatting this message.  The syntax is a standard Java method signature, less return type, qualifiers, semicolon, and extensions. The return type is always <code>String</code> . Argument types can be any Java primitive or class. Classes must be fully qualified if not in <code>java.lang</code> . Classes must also conform to <code>java.text.MessageFormat</code> conventions. In general, class arguments should have a useful <code>toString()</code> method, and the corresponding <code>MessageFormat</code> placeholders must be strings; they must be of the form <code>{n}</code> . Argument names can be any valid name. There can be no more than 10 arguments.  For each argument there must be at least one corresponding placeholder in the <code>messagebody</code> element. Placeholders are of the form <code>{n}</code> , <code>{n,number}</code> or <code>{n,date}</code> .  Example: <code>method="getNoAuthorization(String filename, java.util.Date creDate) "</code>  This example would result in a method in the <code>TextFormatter</code> class as follows:  <code>public String getNoAuthorization(String filename, java.util.Date creDate)</code>	String	No.
retired	Indicates whether message is retired. A retired message is one that was used in a previous release but is now obsolete and not used in the current version. Retired messages are not represented in any generated classes or resource bundles.  Valid values are: <ul style="list-style-type: none"><li>■ true</li><li>■ false</li></ul> Default: false.	String	No.

## Example

The following example shows how to use the message element in log message catalog file:

```
<?xml version="1.0"?>
<!DOCTYPE message_catalog PUBLIC "weblogic-message-catalog-dtd"
    "http://www.bea.com/servers/wls90/dtd/msgcat.dtd">
<message_catalog>
  l10n_package="programs.utils"
  i18n_package="programs.utils"
  subsystem="MYUTIL"
  version="1.0">
  <message messageid="FileMenuTitle">
    <messagebody>
      File
    </messagebody>
  </message>
</message_catalog>
```

## messagebody

Use this element to define a short description for this message.

The messagebody element can contain a 0 to 10 placeholder as {n}, to be replaced by the appropriate argument when the log message is localized.

The message body must include placeholders for all arguments listed in the corresponding method attribute, unless the last argument is throwable or a subclass.

Be careful when using single quotes, because these are specially parsed by `java.text.MessageFormat`. If it is appropriate to quote a message argument, use double quotes (Section , "Example" shows). If a message has one or more placeholders, in order for a single quote to appear correctly (for example, as an apostrophe), it must be followed by a second single quote.

Syntax: a String

### Child Elements

The messagebody element has no child elements.

### Attributes

The messagebody element has no attributes.

### Example

The following example shows how to use the messagebody element in log message catalog file:

```
<?xml version="1.0"?>
<!DOCTYPE message_catalog PUBLIC "weblogic-message-catalog-dtd"
"http://www.bea.com/servers/wls90/dtd/msgcat.dtd">
<message_catalog
  l10n_package="programs.utils"
  i18n_package="programs.utils"
  subsystem="MYUTIL"
  version="1.0"
  baseid="600000"
  endid="600100">
  <logmessage
    messageid="600001"
    severity="warning"
    method="logNoAuthorization(String arg0, java.util.Date arg1,int arg2)">
    <messagebody>
      Could not open file, "{0}" on {1,date} after {2,number} attempts.
    </messagebody>
    <messagedetail>
      The configuration for this application will be defaulted to
      factory settings. Custom configuration information resides
      in file, {0}, created on {1,date}, but is not readable.
    </messagedetail>
    <cause>
      The user is not authorized to use custom configurations. Custom
      configuration information resides in file, {0}, created on
      {1,date}, but is not readable.The attempt has been logged to
      the security log.
    </cause>
    <action>
      The user needs to gain appropriate authorization or learn to
      live with the default settings.
    </action>
  </logmessage>
```

```
</message_catalog>
```

## messagedetail

Use this element to define a detailed description of the event. This element may contain any argument place holders.

Syntax: a String

### Child Elements

The `messagedetail` element supports has no child elements.

### Attributes

The `messagedetail` element has no attributes.

### Example

The following example shows how to use the `messagedetail` element in log message catalog file:

```
<?xml version="1.0"?>
<!DOCTYPE message_catalog PUBLIC "weblogic-message-catalog-dtd"
"http://www.bea.com/servers/wls90/dtd/msgcat.dtd">
<message_catalog
  l10n_package="programs.utils"
  i18n_package="programs.utils"
  subsystem="MYUTIL"
  version="1.0"
  baseid="600000"
  endid="600100">
  <logmessage
    messageid="600001"
    severity="warning"
    method="logNoAuthorization(String arg0, java.util.Date arg1,int arg2)">
    <messagebody>
      Could not open file, {0} on {1,date} after {2,number} attempts.
    </messagebody>
    <messagedetail>
      The configuration for this application will be defaulted to
      factory settings. Custom configuration information resides
      in file, {0}, created on {1,date}, but is not readable.
    </messagedetail>
    <cause>
      The user is not authorized to use custom configurations. Custom
      configuration information resides in file, {0}, created on
      {1,date}, but is not readable.The attempt has been logged to
      the security log.
    </cause>
    <action>
      The user needs to gain appropriate authorization or learn to
      live with the default settings.
    </action>
  </logmessage>
</message_catalog>
```

### cause

Use this element to define the root cause of the problem. This element can contain any argument place holders.

Syntax: a String.

## Child Elements

The cause element supports has no child elements.

## Attributes

The cause element has no attributes.

## Example

The following example shows how to use the cause element in log message catalog file:

```
<?xml version="1.0"?>
<!DOCTYPE message_catalog PUBLIC "weblogic-message-catalog-dtd"
"http://www.bea.com/servers/wls90/dtd/msgcat.dtd">
<message_catalog
  l10n_package="programs.utils"
  i18n_package="programs.utils"
  subsystem="MYUTIL"
  version="1.0"
  baseid="600000"
  endid="600100">
  <logmessage
    messageid="600001"
    severity="warning"
    method="logNoAuthorization(String arg0, java.util.Date arg1,int arg2)">
    <messagebody>
      Could not open file, {0} on {1,date} after {2,number} attempts.
    </messagebody>
    <messagedetail>
      The configuration for this application will be defaulted to
      factory settings. Custom configuration information resides
      in file, {0}, created on {1,date}, but is not readable.
    </messagedetail>
    <cause>
      The user is not authorized to use custom configurations. Custom
      configuration information resides in file, {0}, created on
      {1,date}, but is not readable.The attempt has been logged to
      the security log.
    </cause>
    <action>
      The user needs to gain appropriate authorization or learn to
      live with the default settings.
    </action>
  </logmessage>
</message_catalog>
```

## action

Use this element to define the recommended resolution. This element can contain any argument place holders.

Syntax: a String.

## Child Elements

The action element supports has no child elements.

## Attributes

The action element has no attributes.

## Example

The following example shows how to use the action element in log message catalog file:

```
<?xml version="1.0"?>
<!DOCTYPE message_catalog PUBLIC "weblogic-message-catalog-dtd"
"http://www.bea.com/servers/wls90/dtd/msgcat.dtd">
<message_catalog
  l10n_package="programs.utils"
  i18n_package="programs.utils"
  subsystem="MYUTIL"
  version="1.0"
  baseid="600000"
  endid="600100">
  <logmessage
    messageid="600001"
    severity="warning"
    method="logNoAuthorization(String arg0, java.util.Date arg1,int arg2)">
    <messagebody>
      Could not open file, {0} on {1,date} after {2,number} attempts.
    </messagebody>
    <messagedetail>
      The configuration for this application will be defaulted to
      factory settings. Custom configuration information resides
      in file, {0}, created on {1,date}, but is not readable.
    </messagedetail>
    <cause>
      The user is not authorized to use custom configurations. Custom
      configuration information resides in file, {0}, created on
      {1,date}, but is not readable.The attempt has been logged to
      the security log.
    </cause>
    <action>
      The user needs to gain appropriate authorization or learn to
      live with the default settings.
    </action>
  </logmessage>
</message_catalog>
```



---

---

# Schema Reference: Locale Message Catalog l10n\_msgcat.dtd

This appendix provides a reference to elements of the `l10n_msgcat.dtd` schema, the schema behind XML you use to define log messages localized for a specific Java locale.

This appendix includes the following sections:

- [Overview of the Locale Message Catalog Elements](#)
- [locale\\_message\\_catalog](#)
- [logmessage](#)
- [message](#)
- [messagebody](#)
- [messagedetail](#)
- [cause](#)
- [action](#)

## Overview of the Locale Message Catalog Elements

Oracle Event Processing provides a number of locale message catalog elements that you use to define log messages localized for a given Java locale.

### Element Hierarchy

The top-level Oracle Event Processing locale message catalog elements are organized into the following hierarchies, depending on message catalog type:

- [Example H-1, "Locale-Specific Log Message Catalog Hierarchy"](#)
- [Example H-2, "Locale-Specific Simple Text Catalog Hierarchy"](#)

#### ***Example H-1 Locale-Specific Log Message Catalog Hierarchy***

```
locale_message_catalog
  logmessage
    messagebody
    messagedetail
    cause
    action
```

**Example H-2 Locale-Specific Simple Text Catalog Hierarchy**

```
message_catalog
  message
    messagebody
```

## Examples

This section provides the following message catalog examples:

- [Example H-3, "Locale-Specific Log Message Catalog"](#)
- [Example H-4, "Locale-Specific Simple Text Catalog"](#)

**Example H-3 Locale-Specific Log Message Catalog**

```
<?xml version="1.0"?>
<!DOCTYPE message_catalog PUBLIC
  "weblogic-locale-message-catalog-dtd"
  "http://www.bea.com/servers/wls90/dtd/l10n_msgcat.dtd">
<message_catalog
  l10n_package="programs.utils"
  version="1.0">
  <logmessage
    messageid="600001">
    <messagebody>
      Could not open file, {0} on {1,date} after {2,number} attempts.
    </messagebody>
    <messagedetail>
      The configuration for this application will be defaulted to
      factory settings. Custom configuration information resides
      in file, {0}, created on {1,date}, but is not readable.
    </messagedetail>
    <cause>
      The user is not authorized to use custom configurations. Custom
      configuration information resides in file, {0}, created on
      {1,date}, but is not readable.The attempt has been logged to
      the security log.
    </cause>
    <action>
      The user needs to gain appropriate authorization or learn to
      live with the default settings.
    </action>
  </log_message>
</message_catalog>
```

**Example H-4 Locale-Specific Simple Text Catalog**

```
<?xml version="1.0"?>
<!DOCTYPE message_catalog PUBLIC
  "weblogic-locale-message-catalog-dtd"
  "http://www.bea.com/servers/wls90/dtd/l10n_msgcat.dtd">
<locale_message_catalog
  l10n_package="programs.utils"
  subsystem="MYUTIL"
  version="1.0">
  <message>
    <messageid="FileMenuTitle">
    <messagebody> Fichier </messagebody>
  </message>
</locale_message_catalog>
```

## locale\_message\_catalog

Use this element to define values for a parameterized Oracle CQL or EPL rule in an EPL processor component.

### Child Elements

The locale\_message\_catalog element supports the following child elements:

- [message](#)
- [logmessage](#)

### Attributes

[Table H-1](#) lists the attributes of the locale\_message\_catalog element.

**Table H-1** Attributes of the locale\_message\_catalog Element

Attribute	Description	Data Type	Required?
I10n_package	Java package containing generated LogLocalizer or TextLocalizer properties for this catalog.properties file are named after the catalog file name.  For example, for a French log message catalog called mycat.xml, a properties file called <I10n_package>.mycatLogLocalizer_fr_FR.properties is generated.	String	No.
version	Specifies the version of the I10n_msgcat.dtd being used.  Use: Must be "1.0"  Syntax: x.y where x and y are numeric.  Example: version="1.0"	String	Yes.

### Example

The following example shows how to use the message\_catalog element in log message catalog file:

```
<?xml version="1.0"?>
<!DOCTYPE message_catalog PUBLIC
    "weblogic-locale-message-catalog-dtd"
    "http://www.bea.com/servers/wls90/dtd/I10n_msgcat.dtd">
<message_catalog
    I10n_package="programs.utils"
    version="1.0">
  <logmessage
    messageid="600001">
    <messagebody>
      Could not open file, {0} on {1,date} after {2,number} attempts.
    </messagebody>
    <messagedetail>
      The configuration for this application will be defaulted to
      factory settings. Custom configuration information resides
      in file, {0}, created on {1,date}, but is not readable.
    </messagedetail>
    <cause>
      The user is not authorized to use custom configurations. Custom
      configuration information resides in file, {0}, created on
      {1,date}, but is not readable.The attempt has been logged to
      the security log.
    </cause>
    <action>
      The user needs to gain appropriate authorization or learn to
```

```

        live with the default settings.
    </action>
</logmessage>
</message_catalog>

```

## logmessage

Use this element to define a formal log message.

### Child Elements

The logmessage component configuration element supports the following child elements:

- [messagebody](#)
- [messagedetail](#)
- [cause](#)
- [action](#)

### Attributes

[Table H-2](#) lists the attributes of the logmessage component configuration element.

**Table H-2 Attributes of the logmessage Element**

Attribute	Description	Data Type	Required?
messageid	<p>Unique identifier for this log message. Uniqueness should extend across all catalogs. Value must be in range defined by baseid and endid attributes.</p> <p>Use: Value must be in the range defined by the baseid and endid attributes defined in the message_catalog element.</p> <p>Syntax: one to six decimal digits.</p> <p>Example: messageid="600001"</p>	String	Yes.
datelastchanged	<p>Date/time stamp used for managing modifications to this message. The date is supplied by utilities that run on the catalogs.</p> <p>Use: The date is supplied by utilities that run on the catalogs.</p> <p>Syntax: Long.toString(new Date().getTime());</p>	String	No.

### Example

The following example shows how to use the logmessage element in log message catalog file:

```

<?xml version="1.0"?>
<!DOCTYPE message_catalog PUBLIC
    "weblogic-locale-message-catalog-dtd"
    "http://www.bea.com/servers/wls90/dtd/110n_msgcat.dtd">
<message_catalog
    110n_package="programs.utils"
    version="1.0">
  <logmessage
    messageid="600001">
    <messagebody>
      Could not open file, {0} on {1,date} after {2,number} attempts.
    </messagebody>
    <messagedetail>

```

```

    The configuration for this application will be defaulted to
    factory settings. Custom configuration information resides
    in file, {0}, created on {1,date}, but is not readable.
  </messagedetail>
  <cause>
    The user is not authorized to use custom configurations. Custom
    configuration information resides in file, {0}, created on
    {1,date}, but is not readable.The attempt has been logged to
    the security log.
  </cause>
  <action>
    The user needs to gain appropriate authorization or learn to
    live with the default settings.
  </action>
</logmessage>
</message_catalog>

```

## message

Use this element to define an informal log message.

### Child Elements

The message element supports the following child elements:

- [messagebody](#)

### Attributes

[Table H-3](#) lists the attributes of the message element.

**Table H-3** Attributes of the message Element

Attribute	Description	Data Type	Required?
messageid	Unique identifier for this log message in alpha-numeric string format. Uniqueness is required only within the context of this catalog.	String	Yes.
datelastchanged	Date/time stamp used for managing modifications to this message. The date is supplied by utilities that run on the catalogs.  Use: The date is supplied by utilities that run on the catalogs.  Syntax: <code>Long.toString(new Date().getTime());</code>	String	No.

### Example

The following example shows how to use the message element in log message catalog file:

```

<?xml version="1.0"?>
<!DOCTYPE message_catalog PUBLIC
  "weblogic-locale-message-catalog-dtd"
  "http://www.bea.com/servers/wls90/dtd/l10n_msgcat.dtd">
<locale_message_catalog
  l10n_package="programs.utils"
  version="1.0">
  <message messageid="FileMenuTitle">
    <messagebody>
      File
    </messagebody>
  </message>
</locale_message_catalog>

```

## messagebody

Use this element to define a concise, one-line log message body.

### Child Elements

The messagebody element has no child elements.

### Attributes

The messagebody element has no attributes.

### Example

The following example shows how to use the messagebody element in log message catalog file:

```
<?xml version="1.0"?>
<!DOCTYPE message_catalog PUBLIC
    "weblogic-locale-message-catalog-dtd"
    "http://www.bea.com/servers/wls90/dtd/l10n_msgcat.dtd">
<message_catalog
    l10n_package="programs.utils"
    version="1.0">
  <logmessage
    messageid="600001">
    <messagebody>
      Could not open file, {0} on {1,date} after {2,number} attempts.
    </messagebody>
    <messagedetail>
      The configuration for this application will be defaulted to
      factory settings. Custom configuration information resides
      in file, {0}, created on {1,date}, but is not readable.
    </messagedetail>
    <cause>
      The user is not authorized to use custom configurations. Custom
      configuration information resides in file, {0}, created on
      {1,date}, but is not readable.The attempt has been logged to
      the security log.
    </cause>
    <action>
      The user needs to gain appropriate authorization or learn to
      live with the default settings.
    </action>
  </logmessage>
</message_catalog>
```

## messagedetail

Use this element to define a more detailed explanation of the issue being logged.

### Child Elements

The messagedetail element supports has no child elements.

### Attributes

The messagedetail element has no attributes.

## Example

The following example shows how to use the `messagedetail` element in log message catalog file:

```
<?xml version="1.0"?>
<!DOCTYPE message_catalog PUBLIC
    "weblogic-locale-message-catalog-dtd"
    "http://www.bea.com/servers/wls90/dtd/l10n_msgcat.dtd">
<message_catalog
    l10n_package="programs.utils"
    version="1.0">
  <logmessage
    messageid="600001">
    <messagebody>
      Could not open file, {0} on {1,date} after {2,number} attempts.
    </messagebody>
    <messagedetail>
      The configuration for this application will be defaulted to
      factory settings. Custom configuration information resides
      in file, {0}, created on {1,date}, but is not readable.
    </messagedetail>
    <cause>
      The user is not authorized to use custom configurations. Custom
      configuration information resides in file, {0}, created on
      {1,date}, but is not readable.The attempt has been logged to
      the security log.
    </cause>
    <action>
      The user needs to gain appropriate authorization or learn to
      live with the default settings.
    </action>
  </logmessage>
</message_catalog>
```

## cause

Use this element to define the root cause of the issue being logged.

## Child Elements

The cause element supports has no child elements.

## Attributes

The cause element has no attributes.

## Example

The following example shows how to use the `cause` element in log message catalog file:

```
<?xml version="1.0"?>
<!DOCTYPE message_catalog PUBLIC
    "weblogic-locale-message-catalog-dtd"
    "http://www.bea.com/servers/wls90/dtd/l10n_msgcat.dtd">
<message_catalog
    l10n_package="programs.utils"
    version="1.0">
  <logmessage
    messageid="600001">
    <messagebody>
      Could not open file, {0} on {1,date} after {2,number} attempts.
```

```

</messagebody>
<messagedetail>
  The configuration for this application will be defaulted to
  factory settings. Custom configuration information resides
  in file, {0}, created on {1,date}, but is not readable.
</messagedetail>
<cause>
  The user is not authorized to use custom configurations. Custom
  configuration information resides in file, {0}, created on
  {1,date}, but is not readable.The attempt has been logged to
  the security log.
</cause>
<action>
  The user needs to gain appropriate authorization or learn to
  live with the default settings.
</action>
</logmessage>
</message_catalog>

```

## action

Use this element to define how to fix the issue being logged, if possible.

## Child Elements

The `action` element supports has no child elements.

## Attributes

The `action` element has no attributes.

## Example

The following example shows how to use the `action` element in log message catalog file:

```

<?xml version="1.0"?>
<!DOCTYPE message_catalog PUBLIC
  "weblogic-locale-message-catalog-dtd"
  "http://www.bea.com/servers/wls90/dtd/l10n_msgcat.dtd">
<message_catalog
  l10n_package="programs.utils"
  version="1.0">
  <logmessage
    messageid="600001">
    <messagebody>
      Could not open file, {0} on {1,date} after {2,number} attempts.
    </messagebody>
    <messagedetail>
      The configuration for this application will be defaulted to
      factory settings. Custom configuration information resides
      in file, {0}, created on {1,date}, but is not readable.
    </messagedetail>
    <cause>
      The user is not authorized to use custom configurations. Custom
      configuration information resides in file, {0}, created on
      {1,date}, but is not readable.The attempt has been logged to
      the security log.
    </cause>
    <action>
      The user needs to gain appropriate authorization or learn to
      live with the default settings.
    </action>
  </logmessage>
</message_catalog>

```



```
</logmessage>  
</message_catalog>
```



---

---

# Oracle Event Processing Metadata Annotation Reference

This appendix provides a reference to Oracle Event Processing Java annotations you can use to specify methods that handle adapter lifecycle stages, a method injected with an OSGi service reference, and to configure resource access at design time.

This appendix includes the following sections:

- [Overview of Oracle Event Processing Metadata Annotations](#)
- [com.bea.wlevs.configuration.Activate](#)
- [com.bea.wlevs.configuration.Prepare](#)
- [com.bea.wlevs.configuration.Rollback](#)
- [com.bea.wlevs.util.Service](#)

## Overview of Oracle Event Processing Metadata Annotations

Oracle Event Processing metadata annotations are used to access the configuration of an Oracle Event Processing component. Oracle Event Processing offers the following annotations:

- [Section , "Adapter Lifecycle Annotations"](#)
- [Section , "OSGi Service Reference Annotations"](#)
- [Section , "Resource Access Annotations"](#)

For more information, see:

- [Section , "Oracle Event Processing Application Lifecycle"](#)
- [Section , "Configuring Oracle Event Processing Resource Access"](#)

## 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 lifecycle: when its configuration is prepared, when the configuration is activated, and when the adapter is terminated due to an exception:

- [com.bea.wlevs.configuration.Activate](#)
- [com.bea.wlevs.configuration.Prepare](#)
- [com.bea.wlevs.configuration.Rollback](#)

## 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.

## 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 , "Configuring Oracle Event Processing Resource Access"](#).

## com.bea.wlevs.configuration.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 Event Processing uses to send configuration information to the adapter.

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

This metadata annotation does not have any attributes.

## Example

[Example I-1](#) shows how to use the `@Activate` annotation in the adapter component of the HelloWorld example; only relevant code is shown:

### **Example I-1 @Activate Annotation**

```
package com.bea.wlevs.adapter.example.helloworld;
...
import com.bea.wlevs.configuration.Activate;
import com.bea.wlevs.ede.api.RunnableBean;
import com.bea.wlevs.ede.api.StreamSender;
import com.bea.wlevs.ede.api.StreamSource;
```

```

import com.bea.wlevs.event.example.helloworld.HelloWorldEvent;

public class HelloWorldAdapter implements RunnableBean, StreamSource {
    ...
    @Activate
    public void activateAdapter(HelloWorldAdapterConfig adapterConfig) {
        this.message = adapterConfig.getMessage();
    }
    ...
}

```

The `com.bea.wlevs.adapter.example.helloworld.HelloWorldAdapterConfig` class is a Java representation 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 I-2](#) shows this XSD file also specifies the fully qualified name of the resulting Java configuration object, as shown in bold:

### Example I-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="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>

```

Oracle Event Processing 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.

## com.bea.wlevs.configuration.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 Event Processing uses to send configuration information to the adapter.

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

This metadata annotation does not have any attributes.

## Example

[Example I-3](#), from the adapter component of the HelloWorld example, shows how to use the `@Prepare` annotation; only relevant code is shown:

### Example I-3 @Prepare Annotation

```
package com.bea.wlevs.adapter.example.helloworld;
...
import com.bea.wlevs.configuration.Prepare;
import com.bea.wlevs.ede.api RunnableBean;
import com.bea.wlevs.ede.api StreamSender;
import com.bea.wlevs.ede.api StreamSource;
import com.bea.wlevs.event.example.helloworld.HelloWorldEvent;

public class HelloWorldAdapter implements RunnableBean, StreamSource {
...
    @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 representation of the adapter's configuration XML file; Oracle Event Processing 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 , "com.bea.wlevs.configuration.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.

## com.bea.wlevs.configuration.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 Event Processing uses to send configuration information to the adapter.

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

This metadata annotation does not have any attributes.

## Example

[Example I-4](#), sample code from the adapter component of the HelloWorld example, shows how to use the `@Rollback` annotation; only relevant code is shown:

### **Example I-4 @Rollback Annotation**

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

```

...
import com.bea.wlevs.configuration.Rollback;
import com.bea.wlevs.ede.api.RunnableBean;
import com.bea.wlevs.ede.api.StreamSender;
import com.bea.wlevs.ede.api.StreamSource;
import com.bea.wlevs.event.example.helloworld.HelloWorldEvent;

public class HelloWorldAdapter implements RunnableBean, StreamSource {
...
    @Rollback
    public void rejectConfigurationChange(HelloWorldAdapterConfig adapterConfig) {
    }
}

```

The `com.bea.wlevs.adapter.example.helloworld.HelloWorldAdapterConfig` class is a Java representation of the adapter's configuration XML file; Oracle Event Processing 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, "com.bea.wlevs.configuration.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.

## com.bea.wlevs.util.Service

**Target:** Method

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

## Attributes

[Table I-1](#) describes the attributes of the `com.bea.wlevs.util.Service` JWS annotation.

**Table I-1** Attributes of the `com.bea.wlevs.util.Service` JWS Annotation Tag

Name	Description	Data Type	Required?
<code>serviceName</code>	The name of the bean that backs the injected service. May be null.	String	No.
<code>cardinality</code>	Valid values for this attribute are: <ul style="list-style-type: none"> <li>▪ <code>ServiceCardinality.CO_1</code></li> <li>▪ <code>ServiceCardinality.CO_N</code></li> <li>▪ <code>ServiceCardinality.C1_1</code></li> <li>▪ <code>ServiceCardinality.C1_N</code></li> </ul> Default value is <code>ServiceCardinality.C1_1</code> .	enum	No.
<code>contextClassLoader</code>	Valid values for this attribute are: <ul style="list-style-type: none"> <li>▪ <code>ServiceClassLoader.CLIENT</code></li> <li>▪ <code>ServiceClassLoader.SERVICE_PROVIDER</code></li> <li>▪ <code>ServiceClassLoader.UNMANAGED</code></li> </ul> Default value is <code>ServiceClassLoader.CLIENT</code> .	enum	No.
<code>timeout</code>	Timeout for service resolution in milliseconds. Default value is 30000.	int	No.
<code>serviceType</code>	Interface (or class) of the service to be injected. Default value is <code>Service.class</code> .	Class	No.
<code>filter</code>	Specifies the filter used to narrow service matches. Value may be null.	String	No.



## Example

[Example I-5](#) shows how to use the @Service annotation.

### **Example I-5 @Service Annotation**

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

For another example, see [Section , "Accessing the Event Type Repository"](#).

