

Oracle® Complex Event Processing

Developer's Guide

11g Release 1 (11.1.1.4.0) for Eclipse

E14301-04

January 2011

Oracle Complex Event Processing Developer's Guide 11g Release 1 (11.1.1.4.0) for Eclipse

E14301-04

Copyright © 2007, 2011, Oracle and/or its affiliates. All rights reserved.

Primary Author: Peter Purich

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

U.S. GOVERNMENT RIGHTS Programs, software, databases, and related documentation and technical data delivered to U.S. Government customers are "commercial computer software" or "commercial technical data" pursuant to the applicable Federal Acquisition Regulation and agency-specific supplemental regulations. As such, the use, duplication, disclosure, modification, and adaptation shall be subject to the restrictions and license terms set forth in the applicable Government contract, and, to the extent applicable by the terms of the Government contract, the additional rights set forth in FAR 52.227-19, Commercial Computer Software License (December 2007). Oracle USA, Inc., 500 Oracle Parkway, Redwood City, CA 94065.

This software 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 which may create a risk of personal injury. If you use this software in dangerous applications, then you shall be responsible to take all appropriate fail-safe, backup, redundancy, and other measures to ensure the safe use of this software. Oracle Corporation and its affiliates disclaim any liability for any damages caused by use of this software in dangerous applications.

Oracle is a registered trademark of Oracle Corporation and/or its affiliates. Other names may be trademarks of their respective owners.

This software and documentation may provide access to or information on 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. 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.

Contents

Preface	xlix
Audience	xlix
Documentation Accessibility	xlix
Related Documents	1
Conventions	1

Part I Introduction

1 Overview of Creating Oracle CEP Applications

1.1	Overview of the Oracle CEP Programming Model	1-1
1.1.1	Components of the Oracle CEP Event Processing Network	1-2
1.1.1.1	Adapter	1-2
1.1.1.2	Channel	1-3
1.1.1.3	Processor	1-3
1.1.1.4	Event Bean	1-3
1.1.1.5	Spring Bean	1-3
1.1.1.6	Cache	1-3
1.1.1.7	Table	1-4
1.1.1.8	Nested Stages	1-4
1.1.1.9	Foreign Stages	1-5
1.1.2	Oracle CEP Event Types	1-6
1.1.3	Transmitting Events in the EPN: Stream and Relation Sources and Sinks	1-6
1.1.3.1	Streams and Relations	1-7
1.1.3.2	Stream and Relation Sources	1-7
1.1.3.3	Stream and Relation Sinks	1-8
1.1.3.4	Transmitting Events in the EPN: Examples	1-9
1.1.4	EPN Assembly File	1-11
1.1.5	Component Configuration Files	1-12
1.1.5.1	Accessing Component and Server Configuration Using the ConfigurationPropertyPlaceholderConfigurer Class	1-13
1.1.6	How Components Fit Together	1-14
1.1.7	Extending the EPN	1-14
1.1.8	High Availability and Scalability	1-14
1.1.9	Oracle CEP Application Lifecycle	1-15
1.1.9.1	User Action: Installing an Application or Start the Server With Application Already Deployed	1-16

1.1.9.2	User Action: Suspend Application.....	1-16
1.1.9.3	User Action: Resume Application.....	1-17
1.1.9.4	User Action: Uninstall Application.....	1-17
1.1.9.5	User Action: Update Application.....	1-17
1.1.9.6	User Action: Calling Methods of Stream and Relation Sources and Sinks	1-17
1.1.10	Oracle CEP APIs	1-17
1.2	Oracle CEP IDE for Eclipse.....	1-19
1.3	Creating an Oracle CEP Application.....	1-19
1.4	Configuring Oracle CEP Resource Access	1-21
1.4.1	Static Resource Injection	1-22
1.4.1.1	Static Resource Names.....	1-22
1.4.1.2	Dynamic Resource Names	1-23
1.4.2	Dynamic Resource Injection.....	1-24
1.4.3	Dynamic Resource Lookup Using JNDI.....	1-24
1.4.4	Understanding Resource Name Resolution	1-25
1.5	Next Steps	1-25

2 Overview of Oracle CEP Events

2.1	Oracle CEP Event Types	2-1
2.1.1	Event Type Instantiation and Immutability	2-2
2.1.2	Event Type and Serialization	2-3
2.1.3	Event Type Data Types.....	2-3
2.1.3.1	Event Types Specified as JavaBean or Java Class	2-3
2.1.3.2	Event Types Specified as java.util.Map.....	2-4
2.1.3.3	Event Types Specified as a Tuple.....	2-4
2.1.3.4	Event Types for use With a Database Table Source	2-5
2.1.3.5	Event Types for use With the csvgen Adapter.....	2-5
2.1.4	Creating Oracle CEP Event Types.....	2-5
2.2	Creating an Oracle CEP Event Type as a JavaBean	2-6
2.2.1	How to Create an Oracle CEP Event Type as a JavaBean Using the Event Type Repository Editor 2-6	
2.2.2	How to Create an Oracle CEP Event Type as a JavaBean Manually	2-10
2.3	Creating an Oracle CEP Event Type as a Tuple	2-12
2.3.1	How to Create an Oracle CEP Event Type as a Tuple Using the Event Type Repository Editor 2-12	
2.3.2	How to Create an Oracle CEP Event Type as a Tuple Manually	2-15
2.4	Creating an Oracle CEP Event Type as a Java Class.....	2-17
2.4.1	How to Create an Oracle CEP Event Type as a Java Class Manually.....	2-17
2.5	Creating an Oracle CEP Event Type as a java.util.Map	2-20
2.5.1	How to Create an Oracle CEP Event Type as a java.util.Map	2-20
2.6	Using an Event Type Builder Factory	2-21
2.7	Accessing the Event Type Repository.....	2-22
2.7.1	Using the EPN Assembly File	2-22
2.7.2	Using the Spring-DM @ServiceReference Annotation.....	2-23
2.7.3	Using the Oracle CEP @Service Annotation.....	2-23
2.8	Sharing Event Types Between Application Bundles	2-23

Part II Oracle CEP IDE for Eclipse

3 Overview of the Oracle CEP IDE for Eclipse

3.1	Overview of Oracle CEP IDE for Eclipse.....	3-1
3.1.1	Features	3-1
3.1.2	JDK Requirements	3-2
3.1.3	Default Oracle CEP Domain ocep_domain and Development	3-2
3.2	Installing the Latest Oracle CEP IDE for Eclipse.....	3-2
3.3	Installing the Oracle CEP IDE for Eclipse Distributed With Oracle CEP	3-7
3.4	Configuring Eclipse	3-11

4 Oracle CEP IDE for Eclipse Projects

4.1	Oracle CEP Project Overview.....	4-1
4.2	Creating Oracle CEP Projects.....	4-2
4.2.1	How to Create an Oracle CEP Project.....	4-3
4.3	Creating EPN Assembly Files	4-6
4.3.1	How to Create a New EPN Assembly File Using Oracle CEP IDE for Eclipse	4-7
4.4	Creating Component Configuration Files.....	4-8
4.4.1	How to Create a New Component Configuration File Using Oracle CEP IDE for Eclipse	4-9
4.5	Exporting Oracle CEP Projects.....	4-10
4.5.1	How to Export an Oracle CEP Project	4-10
4.6	Upgrading Projects	4-13
4.6.1	How to Upgrade Projects from Oracle CEP 2.1 to 10.3	4-14
4.6.2	How to Upgrade Projects from Oracle CEP 10.3 to 11g Release 1 (11.1.1).....	4-20
4.7	Managing Libraries and Other Non-Class Files in Oracle CEP Projects	4-29
4.7.1	How to Add a Standard JAR File to an Oracle CEP Project	4-30
4.7.2	How to Add an OSGi Bundle to an Oracle CEP Project	4-35
4.7.3	How to Add a Property File to an Oracle CEP Project.....	4-36
4.7.4	How to Export a Package	4-38
4.7.5	How to Import a Package	4-40
4.8	Configuring Oracle CEP IDE for Eclipse Preferences	4-43
4.8.1	How to Configure Application Library Path Preferences	4-43
4.8.2	How to Configure Problem Severity Preferences	4-43

5 Oracle CEP IDE for Eclipse and Oracle CEP Servers

5.1	Oracle CEP Server Overview	5-1
5.2	Creating Oracle CEP Servers.....	5-3
5.2.1	How to Create a Local Oracle CEP Server and Server Runtime.....	5-3
5.2.2	How to Create a Remote Oracle CEP Server and Server Runtime.....	5-10
5.2.3	How to Create an Oracle CEP Server Runtime	5-16
5.3	Managing Oracle CEP Servers	5-19
5.3.1	How to Start a Local Oracle CEP Server	5-19
5.3.2	How to Stop a Local Oracle CEP Server.....	5-20
5.3.3	How to Attach to an Existing Local Oracle CEP Server Instance	5-21
5.3.4	How to Attach to an Existing Remote Oracle CEP Server Instance	5-22

5.3.5	How to Detach From an Existing Oracle CEP Server Instance	5-22
5.3.6	How to Deploy an Application to an Oracle CEP Server	5-23
5.3.7	How to Configure Connection and Control Settings for Oracle CEP Server	5-26
5.3.8	How to Configure Domain (Runtime) Settings for Oracle CEP Server	5-28
5.3.9	How to Start the Oracle CEP Visualizer from Oracle CEP IDE for Eclipse	5-30
5.4	Debugging an Oracle CEP Application Running on an Oracle CEP Server	5-32
5.4.1	How to Debug an Oracle CEP Application Running on an Oracle CEP Server.....	5-32

6 Oracle CEP IDE for Eclipse and the Event Processing Network

6.1	Opening the EPN Editor	6-1
6.1.1	How to Open the EPN Editor from a Project Folder	6-1
6.1.2	How to Open the EPN Editor from a Context or Configuration File	6-3
6.2	EPN Editor Overview.....	6-4
6.2.1	Flow Representation.....	6-4
6.2.2	Filtering	6-5
6.2.3	Zooming.....	6-6
6.2.4	Layout.....	6-6
6.2.5	Showing and Hiding Unconnected Beans	6-6
6.2.6	Printing and Exporting to an Image	6-7
6.2.7	Configuration Badging	6-7
6.2.8	Link Specification Location Indicator	6-8
6.2.9	Nested Stages	6-9
6.2.10	Event Type Repository Editor.....	6-10
6.3	Navigating the EPN Editor.....	6-11
6.3.1	Moving the Canvas.....	6-11
6.3.2	Shortcuts to Component Configuration and EPN Assembly Files	6-11
6.3.3	Hyperlinking	6-12
6.3.3.1	Hyperlinking in Component Configuration and EPN Assembly Files.....	6-12
6.3.3.2	Hyperlinking in Oracle CQL Statements	6-13
6.3.4	Context Menus	6-14
6.3.5	Browsing Oracle CEP Types	6-15
6.3.5.1	How to Browse Oracle CEP Types	6-15
6.4	Using the EPN Editor	6-18
6.4.1	Creating Nodes	6-18
6.4.1.1	How to Create a Basic Node	6-19
6.4.1.2	How to Create an Adapter Node	6-21
6.4.1.3	How to Create a Processor Node	6-26
6.4.2	Connecting Nodes	6-28
6.4.2.1	How to Connect Nodes	6-28
6.4.3	Laying Out Nodes	6-30
6.4.4	Renaming Nodes.....	6-30
6.4.5	Deleting Nodes.....	6-30

Part III Building the Oracle CEP Event Processing Network

7 Configuring JMS Adapters

7.1	Overview of JMS Adapter Configuration	7-1
7.1.1	JMS Service Providers	7-1
7.1.2	Inbound JMS Adapter	7-2
7.1.2.1	Conversion Between JMS Messages and Event Types.....	7-2
7.1.2.2	Single and Multi-threaded Inbound JMS Adapters	7-3
7.1.3	Outbound JMS Adapter	7-3
7.2	Configuring a JMS Adapter for a JMS Service Provider	7-4
7.2.1	How to Configure a JMS Adapter Using the Oracle CEP IDE for Eclipse	7-4
7.2.2	How to Configure a JMS Adapter Manually	7-5
7.2.3	How to Configure a JMS Adapter for Oracle WebLogic Server JMS Manually	7-7
7.2.4	How to Configure a JMS Adapter for Tibco EMS JMS Manually	7-10
7.3	Creating a Custom Converter Between JMS Messages and Event Types	7-12
7.3.1	How to Create a Custom Converter for the Inbound JMS Adapter.....	7-12
7.3.2	How to Create a Custom Converter for the Outbound JMS Adapter	7-14
7.4	Encrypting Passwords in the JMS Adapter Component Configuration File	7-15
7.4.1	How to Encrypt Passwords in the JMS Adapter Component Configuration File ..	7-15
7.5	Configuring the JMS Adapter EPN Assembly File	7-16
7.5.1	JMS Inbound Adapter EPN Assembly File Configuration.....	7-17
7.5.2	JMS Outbound Adapter EPN Assembly File Configuration.....	7-18
7.6	Configuring the JMS Adapter Component Configuration File.....	7-19
7.6.1	JMS Inbound Adapter Component Configuration.....	7-19
7.6.2	JMS Outbound Adapter Component Configuration.....	7-22

8 Configuring HTTP Publish-Subscribe Server Adapters

8.1	Overview of HTTP Publish-Subscribe Server Adapter Configuration.....	8-1
8.1.1	Overview of the Built-In Pub-Sub Adapter for Publishing	8-2
8.1.1.1	Local Publishing	8-2
8.1.1.2	Remote Publishing	8-3
8.1.2	Overview of the Built-In Pub-Sub Adapter for Subscribing	8-4
8.1.3	Converting Between JSON Messages and Event Types	8-5
8.2	Configuring an HTTP Pub-Sub Adapter	8-5
8.2.1	How to Configure an HTTP Pub-Sub Adapter Using the Oracle CEP IDE for Eclipse	8-5
8.2.2	How to Configure an HTTP Pub-Sub Adapter Manually	8-6
8.3	Creating a Custom Converter Between the HTTP Pub-Sub Messages and Event Types	8-8
8.4	Configuring the HTTP Pub-Sub Adapter EPN Assembly File	8-10
8.4.1	HTTP Pub-Sub Adapter for Publishing EPN Assembly File Configuration.....	8-10
8.4.2	HTTP Pub-Sub Adapter for Subscribing EPN Assembly File Configuration.....	8-12
8.5	Configuring the HTTP Pub-Sub Adapter Component Configuration File	8-13
8.5.1	HTTP Pub-Sub Adapter for Publishing Component Configuration.....	8-13
8.5.2	HTTP Pub-Sub Adapter for Subscribing Component Configuration.....	8-15

9 Configuring Channels

9.1	Overview of Channel Configuration	9-1
9.1.1	When to Use a Channel.....	9-2

9.1.2	Channels Representing Streams and Relations	9-3
9.1.2.1	Channels as Streams.....	9-3
9.1.2.2	Channels as Relations	9-3
9.1.3	System-Timestamped Channels	9-4
9.1.4	Application-Timestamped Channels.....	9-4
9.1.5	Controlling Which Queries Output to a Downstream Channel: selector.....	9-4
9.1.6	Batch Processing Channels.....	9-6
9.1.7	EventPartitioner Channels	9-6
9.2	Configuring a Channel.....	9-6
9.2.1	How to Configure a System-Timestamped Channel Using Oracle CEP IDE for Eclipse .	9-6
9.2.2	How to Configure an Application-Timestamped Channel Using Oracle CEP IDE for Eclipse	9-10
9.2.3	How to Create a Channel Component Configuration File Manually	9-13
9.3	Example Channel Configuration Files.....	9-16
9.3.1	Channel Component Configuration File.....	9-16
9.3.2	Channel EPN Assembly File	9-17

10 Configuring Oracle CQL Processors

10.1	Overview of Oracle CQL Processor Configuration	10-1
10.1.1	Controlling Which Queries Output to a Downstream Channel.....	10-3
10.2	Configuring an Oracle CQL Processor	10-3
10.2.1	How to Configure an Oracle CQL Processor Using Oracle CEP IDE for Eclipse ...	10-3
10.2.2	How to Create an Oracle CQL Processor Component Configuration File Manually	10-4
10.3	Configuring an Oracle CQL Processor Table Source.....	10-7
10.3.1	How to Configure an Oracle CQL Processor Table Source Using Oracle CEP IDE for Eclipse	10-7
10.4	Configuring an Oracle CQL Processor Cache Source.....	10-11
10.5	Example Oracle CQL Processor Configuration Files.....	10-11
10.5.1	Oracle CQL Processor Component Configuration File.....	10-11
10.5.2	Oracle CQL Processor EPN Assembly File	10-12

11 Configuring EPL Processors

11.1	Overview of EPL Processor Component Configuration.....	11-1
11.2	Configuring an EPL Processor	11-3
11.2.1	How to Configure an EPL Processor Manually	11-3
11.3	Configuring an EPL Processor Cache Source	11-5
11.4	Example EPL Processor Configuration Files	11-6
11.4.1	EPL Processor Component Configuration File	11-6
11.4.2	EPL Processor EPN Assembly File.....	11-6

12 Configuring Caching

12.1	Overview of Oracle CEP Cache Configuration	12-1
12.1.1	Caching Use Cases.....	12-4
12.1.1.1	Use Case: Publishing Events to a Cache	12-4
12.1.1.2	Use Case: Consuming Data From a Cache	12-4

12.1.1.3	Use Case: Updating and Deleting Data in a Cache	12-4
12.1.1.4	Use Case: Using a Cache in a Multi-Server Domain	12-5
12.1.2	Additional Caching Features	12-5
12.1.3	Caching APIs	12-5
12.2	Configuring an Oracle CEP Local Caching System and Cache	12-6
12.2.1	Configuring an Oracle CEP Local Cache as an Event Listener	12-11
12.2.1.1	Specifying the Key Used to Index an Oracle CEP Local Cache	12-11
12.2.1.1.1	Specifying a Key Property in EPN Assembly File	12-12
12.2.1.1.2	Using a Metadata Annotation to Specify a Key	12-12
12.2.1.1.3	Specifying a Composite Key	12-13
12.2.2	Configuring an Oracle CEP Local Cache as an Event Source	12-13
12.2.3	Configuring an Oracle CEP Local Cache Loader	12-13
12.2.4	Configuring an Oracle CEP Local Cache Store	12-14
12.3	Configuring an Oracle Coherence Caching System and Cache	12-14
12.3.1	Configuring the Oracle Coherence Caching System and Caches	12-17
12.3.1.1	The coherence-cache-config.xml File	12-17
12.3.1.2	The tangosol-coherence-override.xml File	12-19
12.3.2	Configuring an Oracle Coherence Cache as an Event Listener	12-20
12.3.2.1	Specifying the Key Used to Index an Oracle Coherence Cache	12-20
12.3.2.1.1	Specifying a Key Property in EPN Assembly File	12-20
12.3.2.1.2	Using a Metadata Annotation to Specify a Key	12-21
12.3.2.1.3	Specifying a Composite Key	12-21
12.3.3	Configuring an Oracle Coherence Cache as an Event Source	12-22
12.3.4	Configuring an Oracle Coherence Cache Loader or Store	12-22
12.3.4.1	Configuring an Oracle Coherence Cache Loader	12-23
12.3.4.2	Configuring an Oracle Coherence Cache Store	12-24
12.3.5	Configuring a Shared Oracle Coherence Cache	12-25
12.4	Configuring a Third-Party Caching System and Cache	12-25
12.5	Accessing a Cache From an Oracle CQL Statement	12-28
12.5.1	How to Access a Cache From an Oracle CQL Statement	12-30
12.6	Accessing a Cache From an EPL Statement	12-31
12.6.1	How To Access a Cache From an EPL Statement	12-32
12.7	Accessing a Cache From an Adapter	12-33
12.8	Accessing a Cache From a Business POJO	12-34
12.9	Accessing a Cache From an Oracle CQL User-Defined Function	12-34
12.10	Accessing a Cache From an EPL User-Defined Function	12-35
12.11	Accessing a Cache Using JMX	12-36
12.11.1	How to Access a Cache With JMX Using Oracle CEP Visualizer	12-37
12.11.2	How to Access a Cache With JMX Using Java	12-37

13 Configuring Event Record and Playback

13.1	Overview of Configuring Event Record and Playback	13-1
13.1.1	Storing Events in the Persistent Event Store	13-2
13.1.1.1	Default Persistent Event Store	13-2
13.1.1.2	Custom Persistent Event Store	13-2
13.1.1.3	Persistent Event Store Schema	13-2
13.1.2	Recording Events	13-3

13.1.3	Playing Back Events	13-3
13.1.4	Querying Stored Events.....	13-3
13.1.5	Record and Playback Example	13-4
13.2	Configuring Event Record and Playback in Your Application.....	13-4
13.2.1	Configuring an Event Store for Oracle CEP Server	13-5
13.2.2	Configuring a Component to Record Events	13-5
13.2.3	Configuring a Component to Playback Events	13-8
13.2.4	Starting and Stopping the Record and Playback of Events	13-10
13.2.5	Description of the Berkeley Database Schema	13-11
13.3	Creating a Custom Event Store Provider	13-12

Part IV Extending the Oracle CEP Event Processing Network

14 Configuring Custom Adapters

14.1	Overview of Custom Adapters.....	14-1
14.1.1	Custom Adapter Event Sources and Event Sinks	14-2
14.1.1.1	Custom Adapters as Event Sources	14-2
14.1.1.2	Custom Adapters as Event Sinks	14-2
14.1.2	Custom Adapter Factories.....	14-3
14.1.3	Single and Multi-threaded Adapters	14-3
14.2	Implementing a Custom Adapter.....	14-3
14.2.1	How to Implement a Custom Adapter Using Ant	14-3
14.2.2	How to Implement a Custom Adapter Manually.....	14-4
14.2.2.1	Implementing a Custom Adapter as an Event Source	14-5
14.2.2.2	Implementing a Custom Adapter as an Event Sink	14-8
14.2.3	Implementing a Custom Adapter Factory	14-11
14.3	Passing Login Credentials from an Adapter to a Data Feed Provider.....	14-12
14.3.1	How to Pass Static Login Credentials to the Data Feed Provider	14-12
14.3.2	How to Pass Dynamic Login Credentials to the Data Feed Provider.....	14-13
14.3.3	How to Access Login Credentials From an Adapter at Runtime	14-15
14.4	Configuring the Custom Adapter EPN Assembly File	14-16
14.4.1	Registering the Custom Adapter Factory	14-16
14.4.2	Declaring the Custom Adapter Components in your Application	14-17
14.5	Configuring the Custom Adapter Component Configuration File.....	14-17
14.5.1	How to Configure a Custom Adapter Manually	14-18
14.5.1.1	Example of a Custom Adapter Configuration File.....	14-18

15 Configuring Custom Event Beans

15.1	Overview of Custom Event Beans.....	15-1
15.1.1	Custom Event Bean Event Sources and Event Sinks	15-1
15.1.1.1	Custom Event Beans as Event Sources	15-1
15.1.1.2	Custom Event Beans as Event Sinks	15-2
15.1.2	Custom Event Bean Factories.....	15-2
15.2	Implementing a Custom Event Bean	15-2
15.2.1	Implementing a Custom Event Bean as an Event Source	15-3
15.2.2	Implementing a Custom Event Bean as an Event Sink	15-4

15.2.3	Implementing a Custom Event Bean Factory	15-6
15.3	Configuring the Custom Event Bean EPN Assembly File	15-6
15.3.1	Registering the Custom Event Bean Factory	15-6
15.3.2	Declaring the Custom Event Bean Components in your Application.....	15-7
15.4	Configuring the Custom Event Bean Component Configuration File.....	15-7
15.4.1	How to Configure a Custom Event Bean Manually	15-8
15.4.1.1	Example of a Custom Event Bean Configuration File.....	15-9
16	Configuring Custom Spring Beans	
16.1	Overview of Custom Spring Beans	16-1
16.1.1	Spring Bean Event Sources and Event Sinks	16-1
16.1.1.1	Spring Beans as Event Sources	16-1
16.1.1.2	Spring Beans as Event Sinks	16-2
16.2	Implementing a Custom Spring Bean.....	16-2
16.2.1	Implementing a Custom Spring Bean as an Event Source	16-3
16.2.2	Implementing a Custom Spring Bean as an Event Sink.....	16-4
16.3	Configuring the Custom Spring Bean EPN File	16-5
16.3.1	Declaring the Custom Spring Bean Components in your Application	16-5
17	Configuring Web Services	
17.1	Understanding Oracle CEP and Web Services.....	17-1
17.2	How to Invoke a Web Service From an Oracle CEP Application.....	17-1
17.3	How to Expose an Oracle CEP Application as a Web Service	17-2
18	Configuring Applications With Data Cartridges	
18.1	Understanding Data Cartridge Application Context	18-1
18.2	How to Configure Oracle Spatial Application Context	18-1
18.3	How to Configure Oracle JDBC Data Cartridge Application Context	18-3
19	Extending Component Configuration	
19.1	Overview of Extending Component Configuration.....	19-1
19.1.1	Extending Component Configuration Using Annotations.....	19-2
19.1.2	Extending Component Configuration Using an XSD	19-2
19.2	Extending Component Configuration	19-2
19.2.1	How to Extend Component Configuration Using Annotations.....	19-2
19.2.2	How to Extend Component Configuration Using an XSD.....	19-4
19.2.2.1	Creating the XSD Schema File	19-6
19.2.2.1.1	Complete Example of an Extended XSD Schema File	19-8
19.3	Programming Access to the Configuration of a Custom Adapter or Event Bean.....	19-9
19.3.1	How to Access Component Configuration Using Resource Injection.....	19-9
19.3.2	How to Access Component Configuration Using Lifecycle Callbacks.....	19-10
19.3.2.1	Lifecycle Callback Annotations	19-11
19.3.2.2	Lifecycle	19-11

Part V Developing Applications for High Availability

20 Understanding High Availability

20.1	High Availability Architecture	20-1
20.1.1	High Availability Lifecycle and Failover	20-2
20.1.1.1	Secondary Failure	20-3
20.1.1.2	Primary Failure and Failover	20-3
20.1.1.3	Rejoining the High Availability Multi-Server Domain	20-3
20.1.2	Deployment Group and Notification Group	20-4
20.1.3	High Availability Components.....	20-4
20.1.3.1	High Availability Input Adapter.....	20-6
20.1.3.2	Buffering Output Adapter.....	20-6
20.1.3.3	Broadcast Output Adapter	20-6
20.1.3.4	Correlating Output Adapter	20-7
20.1.3.5	ActiveActiveGroupBean.....	20-7
20.1.4	High Availability and Scalability	20-7
20.1.5	High Availability and Oracle Coherence	20-8
20.2	Choosing a Quality of Service	20-9
20.2.1	Simple Failover	20-9
20.2.2	Simple Failover with Buffering.....	20-10
20.2.3	Light-Weight Queue Trimming.....	20-10
20.2.4	Precise Recovery with JMS	20-11
20.3	Designing an Oracle CEP Application for High Availability.....	20-12
20.3.1	Primary Oracle CEP High Availability Use Case	20-12
20.3.2	High Availability Design Patterns	20-13
20.3.2.1	Select the Minimum High Availability Your Application can Tolerate.....	20-13
20.3.2.2	Use Oracle CEP High Availability Components at All Ingress and Egress Points	20-13
20.3.2.3	Only Preserve What You Need	20-14
20.3.2.4	Limit Oracle CEP Application State.....	20-14
20.3.2.5	Choose an Adequate warm-up-window Time	20-14
20.3.2.5.1	Type 1 Applications	20-14
20.3.2.5.2	Type 2 Applications	20-15
20.3.2.6	Ensure Applications are Idempotent.....	20-15
20.3.2.7	Source Event Identity Externally.....	20-16
20.3.2.8	Understand the Importance of Event Ordering	20-16
20.3.2.8.1	Prefer Deterministic Behavior	20-16
20.3.2.8.2	Avoid Multithreading.....	20-16
20.3.2.8.3	Prefer Monotonic Event Identifiers	20-17
20.3.2.9	Write Oracle CQL Queries with High Availability in Mind	20-17
20.3.2.10	Avoid Coupling Servers	20-17
20.3.2.11	Plan for Server Recovery	20-17
20.3.3	Oracle CQL Query Restrictions	20-17
20.3.3.1	Range-Based Windows	20-18
20.3.3.2	Tuple-Based Windows.....	20-18
20.3.3.3	Partitioned Windows	20-18
20.3.3.4	Sliding Windows	20-18
20.3.3.5	DURATION Clause and Non-Event Detection.....	20-19
20.3.3.6	Prefer Application Time	20-19

21 Configuring High Availability

21.1	Configuring High Availability Quality of Service	21-1
21.1.1	How to Configure Simple Failover	21-1
21.1.2	How to Configure Simple Failover With Buffering.....	21-5
21.1.3	How to Configure Light-Weight Queue Trimming	21-9
21.1.4	How to Configure Precise Recovery With JMS.....	21-16
21.2	Configuring High Availability Adapters	21-25
21.2.1	How to Configure the High Availability Input Adapter.....	21-25
21.2.1.1	High Availability Input Adapter EPN Assembly File Configuration	21-26
21.2.1.2	High Availability Input Adapter Component Configuration File Configuration	21-27
21.2.2	How to Configure the Buffering Output Adapter	21-28
21.2.2.1	Buffering Output Adapter EPN Assembly File Configuration	21-28
21.2.2.2	Buffering Output Adapter Component Configuration File Configuration	21-29
21.2.3	How to Configure the Broadcast Output Adapter	21-29
21.2.3.1	Broadcast Output Adapter EPN Assembly File Configuration.....	21-29
21.2.3.2	Broadcast Output Adapter Component Configuration File Configuration	21-30
21.2.4	How to Configure the Correlating Output Adapter	21-31
21.2.4.1	Correlating Output Adapter EPN Assembly File Configuration.....	21-31
21.2.4.2	Correlating Output Adapter Component Configuration File Configuration	21-32

Part VI Developing Applications for Scalability

22 Understanding Scalability

22.1	Scalability Options	22-1
22.1.1	Scalability and High Availability	22-1
22.2	Scalability Components	22-2
22.2.1	EventPartitioner	22-2
22.2.1.1	EventPartitioner Implementation	22-2
22.2.1.2	EventPartitioner Load Balancing	22-3
22.2.1.3	EventPartitioner Initialization	22-3
22.2.1.4	EventPartitioner Threading	22-3
22.2.1.5	EventPartitioner Restrictions	22-3
22.2.2	ActiveActiveGroupBean.....	22-4
22.2.2.1	Scalability in an Oracle CEP Application Using the ActiveActiveGroupBean Without High Availability	22-4
22.2.2.2	Scalability in an Oracle CEP Application Using the ActiveActiveGroupBean With High Availability	22-6

23 Configuring Scalability

23.1	Configuring Scalability With a Channel EventPartitioner.....	23-1
23.1.1	How to Configure Scalability With the Default Channel EventPartitioner	23-2
23.1.2	How to Configure Scalability With a Custom Channel EventPartitioner	23-4
23.2	Configuring Scalability With the ActiveActiveGroupBean.....	23-8
23.2.1	How to Configure Scalability in a JMS Application Without Oracle CEP High Availability	23-9

23.2.2	How to Configure Scalability in a JMS Application With Oracle CEP High Availability	23-11
23.2.3	How to Configure the ActiveActiveGroupBean Group Pattern Match	23-17

Part VII Assembly, Deployment, and Testing

24 Assembling and Deploying Oracle CEP Applications

24.1	Overview of Application Assembly and Deployment.....	24-1
24.1.1	Applications.....	24-1
24.1.2	Application Dependencies	24-2
24.1.2.1	Private Application Dependencies.....	24-2
24.1.2.2	Shared Application Dependencies.....	24-2
24.1.2.3	Native Code Dependencies.....	24-3
24.1.3	Application Libraries.....	24-3
24.1.3.1	Library Directory	24-4
24.1.3.2	Library Extensions Directory	24-4
24.1.3.3	Creating Application Libraries.....	24-5
24.1.4	Deployment and Deployment Order.....	24-5
24.1.5	Configuration History Management	24-6
24.2	Assembling an Oracle CEP Application.....	24-6
24.2.1	Assembling an Oracle CEP Application Using Oracle CEP IDE for Eclipse	24-6
24.2.2	Assembling an Oracle CEP Application Manually.....	24-7
24.2.2.1	Creating the MANIFEST.MF File.....	24-8
24.2.2.2	Accessing Third-Party JAR Files	24-10
24.2.2.2.1	Accessing Third-Party JAR Files Using Bundle-Classpath	24-10
24.2.2.2.2	Accessing Third-Party JAR Files Using -Xbootclasspath	24-10
24.2.3	Assembling Applications With Foreign Stages.....	24-11
24.2.4	Assembling a Custom Adapter or Event Bean in Its Own Bundle	24-12
24.2.4.1	How to Assemble a Custom Adapter in its Own Bundle.....	24-12
24.2.4.2	How to Assemble a Custom Event Bean in its Own Bundle	24-13
24.3	Managing Application Libraries.....	24-13
24.3.1	How to Define the Application Library Directory Using Oracle CEP IDE for Eclipse	24-14
24.3.1.1	How to Configure an Absolute Path	24-15
24.3.1.2	How to Extend a Path Variable	24-16
24.3.2	How to Create an Application Library Using bundler.sh	24-18
24.3.3	How to Create an Application Library Using Oracle CEP IDE for Eclipse	24-21
24.3.4	How to Update an Application Library Using Oracle CEP IDE for Eclipse.....	24-30
24.3.5	How to View an Application Library Using the Oracle CEP Visualizer.....	24-31
24.4	Managing Log Message Catalogs.....	24-31
24.4.1	Using Message Catalogs With Oracle CEP Server.....	24-32
24.4.1.1	Message Catalog Hierarchy	24-33
24.4.1.2	Guidelines for Naming Message Catalogs.....	24-33
24.4.1.3	Using Message Arguments	24-33
24.4.1.4	Message Catalog Formats.....	24-34
24.4.1.4.1	Log Message Catalog	24-34
24.4.1.4.2	Simple Text Catalog	24-35

24.4.1.4.3	Locale-Specific Catalog.....	24-36
24.4.1.5	Message Catalog Localization	24-36
24.4.2	How to Parse a Message Catalog to Generate Logger and TextFormatter Classes for Localization 24-37	
24.5	Deploying Oracle CEP Applications.....	24-38
24.5.1	How to Deploy an Oracle CEP Application Using Oracle CEP IDE for Eclipse...	24-39
24.5.2	How to Deploy an Oracle CEP Application Using Oracle CEP Visualizer.....	24-39
24.5.3	How to Deploy an Oracle CEP Application Using the Deployer Utility	24-39

25 Testing Applications With the Load Generator and csvgen Adapter

25.1	Overview of Testing Applications With the Load Generator and csvgen Adapter.....	25-1
25.2	Configuring and Running the Load Generator Utility.....	25-1
25.3	Creating a Load Generator Property File	25-2
25.4	Creating a Data Feed File.....	25-3
25.5	Configuring the csvgen Adapter in Your Application.....	25-4

26 Testing Applications With the Event Inspector

26.1	Overview of Testing Applications With the Event Inspector	26-1
26.1.1	Tracing Events.....	26-1
26.1.2	Injecting Events.....	26-2
26.1.3	Event Inspector Event Types.....	26-2
26.1.4	Event Inspector HTTP Publish-Subscribe Channel and Server	26-3
26.1.5	Event Inspector Clients	26-4
26.1.5.1	Oracle CEP Visualizer	26-4
26.2	Configuring the Event Inspector HTTP Pub-Sub Server	26-4
26.2.1	How to Configure a Local Event Inspector HTTP Pub-Sub Server	26-5
26.2.2	How to Configure a Remote Event Inspector HTTP Pub-Sub Server.....	26-5
26.3	Injecting Events	26-6
26.3.1	How to Inject Events Using Oracle CEP Visualizer.....	26-7
26.4	Tracing Events	26-7
26.4.1	How to Trace Events Using Oracle CEP Visualizer	26-7

27 Performance Tuning

27.1	EPN Performance Tuning.....	27-1
27.1.1	Event Partitioner Channel	27-1
27.1.2	Batching Channel.....	27-1
27.1.3	Scalability Using the ActiveActiveGroupBean.....	27-1
27.2	High Availability Performance Tuning	27-2
27.2.1	Host Configuration.....	27-2
27.2.2	High Availability Input Adapter and Quality of Service	27-2
27.2.3	High Availability Input Adapter Configuration.....	27-2
27.2.4	Broadcast Output Adapter Configuration	27-2
27.2.5	Oracle Coherence Performance Tuning Options	27-3
27.2.5.1	Oracle Coherence Heartbeat Frequency.....	27-3
27.2.5.2	Oracle Coherence Serialization.....	27-3

Part VIII Oracle CEP Reference

A Additional Information about Spring and OSGi

B Oracle CEP Schemas

B.1	EPN Assembly Schema spring-wlevs-v11_1_1_3.xsd	B-1
B.1.1	Example EPN Assembly File.....	B-1
B.2	Component Configuration Schema wlevs_application_config.xsd	B-2
B.2.1	Example Component Configuration File	B-2
B.3	Deployment Schema deployment.xsd	B-3
B.3.1	Example Deployment XML File	B-3
B.4	Server Configuration Schema wlevs_server_config.xsd	B-3
B.4.1	Example Server Configuration XML File.....	B-4

C Schema Reference: EPN Assembly spring-wlevs-v11_1_1_3.xsd

C.1	Overview of the Oracle CEP Application Assembly Elements.....	C-1
C.1.1	Element Hierarchy	C-1
C.1.2	Example of an EPN Assembly File That Uses Oracle CEP Elements.....	C-2
C.2	wlevs:adapter	C-3
C.2.1	Child Elements	C-3
C.2.2	Attributes	C-3
C.2.3	Example.....	C-5
C.3	wlevs:application-timestamped.....	C-5
C.3.1	Child Elements	C-5
C.3.2	Attributes	C-5
C.3.3	Example.....	C-6
C.4	wlevs:cache	C-6
C.4.1	Child Elements	C-6
C.4.2	Attributes	C-6
C.4.3	Example.....	C-7
C.5	wlevs:cache-listener	C-7
C.5.1	Attributes	C-8
C.5.2	Example.....	C-8
C.6	wlevs:cache-loader.....	C-8
C.6.1	Attributes	C-8
C.6.2	Example.....	C-8
C.7	wlevs:cache-source	C-9
C.7.1	Attributes	C-9
C.7.2	Example.....	C-9
C.8	wlevs:cache-store	C-10
C.8.1	Attributes	C-10
C.8.2	Example.....	C-10
C.9	wlevs:caching-system.....	C-10
C.9.1	Child Elements	C-10
C.9.2	Attributes	C-10
C.9.3	Example.....	C-11

C.10	wlevs:channel	C-11
C.10.1	Child Elements	C-12
C.10.2	Attributes	C-12
C.10.3	Example.....	C-13
C.11	wlevs:event-bean.....	C-13
C.11.1	Child Elements	C-14
C.11.2	Attributes	C-14
C.11.3	Example.....	C-15
C.12	wlevs:event-type-repository.....	C-15
C.12.1	Child Elements	C-15
C.12.2	Example.....	C-15
C.13	wlevs:event-type	C-16
C.13.1	Child Elements	C-16
C.13.2	Attributes	C-16
C.13.3	Example.....	C-17
C.14	wlevs:expression	C-17
C.14.1	Example.....	C-17
C.15	wlevs:factory.....	C-17
C.15.1	Attributes	C-18
C.15.2	Example.....	C-18
C.16	wlevs:function	C-18
C.16.1	Attributes	C-19
C.16.2	Example.....	C-19
C.16.2.1	Single-Row User-Defined Function on an Oracle CQL Processor	C-19
C.16.2.2	Single-Row User-Defined Function on an EPL Processor.....	C-20
C.16.2.3	Aggregate User-Defined Function on an Oracle CQL Processor	C-21
C.16.2.4	Aggregate User-Defined Function on an EPL Processor.....	C-22
C.16.2.5	Specifying the Implementation Class: Nested Bean or Reference.....	C-24
C.17	wlevs:instance-property.....	C-25
C.17.1	Child Elements	C-25
C.17.2	Attributes	C-25
C.17.3	Example.....	C-26
C.18	wlevs:listener	C-26
C.18.1	Attributes	C-26
C.18.2	Example.....	C-27
C.19	wlevs:metadata.....	C-27
C.19.1	Child Elements	C-27
C.19.2	Attributes	C-27
C.19.3	Example.....	C-27
C.20	wlevs:processor	C-28
C.20.1	Child Elements	C-28
C.20.2	Attributes	C-28
C.20.3	Example.....	C-29
C.21	wlevs:property	C-29
C.21.1	Child Elements	C-29
C.21.2	Attributes	C-29
C.21.3	Example.....	C-30

C.22	wlevs:source.....	C-30
C.22.1	Attributes	C-30
C.22.2	Example.....	C-31
C.23	wlevs:table.....	C-31
C.23.1	Attributes	C-31
C.23.2	Example.....	C-31
C.24	wlevs:table-source.....	C-31
C.24.1	Attributes	C-32
C.24.2	Example.....	C-32

D Schema Reference: Component Configuration wlevs_application_config.xsd

D.1	Overview of the Oracle CEP Component Configuration Elements	D-1
D.1.1	Element Hierarchy	D-1
D.1.2	Example of an Oracle CEP Component Configuration File	D-10
D.2	accept-backlog	D-11
D.2.1	Child Elements	D-11
D.2.2	Attributes	D-11
D.2.3	Example.....	D-11
D.3	adapter	D-11
D.3.1	Child Elements	D-11
D.3.2	Attributes	D-12
D.3.3	Example.....	D-12
D.4	amount.....	D-12
D.4.1	Child Elements	D-12
D.4.2	Attributes	D-12
D.4.3	Example.....	D-12
D.5	application.....	D-13
D.5.1	Child Elements	D-13
D.5.2	Attributes	D-13
D.5.3	Example.....	D-13
D.6	average-interval.....	D-13
D.6.1	Child Elements	D-14
D.6.2	Attributes	D-14
D.6.3	Example.....	D-14
D.7	average-latency.....	D-14
D.7.1	Child Elements	D-14
D.7.2	Attributes	D-15
D.7.3	Example.....	D-15
D.8	batch-size.....	D-15
D.8.1	Child Elements	D-15
D.8.2	Attributes	D-15
D.8.3	Example.....	D-15
D.9	batch-time-out	D-16
D.9.1	Child Elements	D-16
D.9.2	Attributes	D-16
D.9.3	Example.....	D-16
D.10	binding.....	D-16

D.10.1	Child Elements	D-16
D.10.2	Attributes	D-17
D.10.3	Example.....	D-17
D.11	bindings (jms-adapter)	D-17
D.11.1	Child Elements	D-17
D.11.2	Attributes	D-18
D.11.3	Example.....	D-18
D.12	bindings (processor)	D-18
D.12.1	Child Elements	D-18
D.12.2	Attributes	D-19
D.12.3	Example.....	D-19
D.13	buffer-size.....	D-19
D.13.1	Child Elements	D-19
D.13.2	Attributes	D-19
D.13.3	Example.....	D-19
D.14	buffer-write-attempts	D-20
D.14.1	Child Elements	D-20
D.14.2	Attributes	D-20
D.14.3	Example.....	D-20
D.15	buffer-write-timeout.....	D-21
D.15.1	Child Elements	D-21
D.15.2	Attributes	D-21
D.15.3	Example.....	D-21
D.16	cache.....	D-21
D.16.1	Child Elements	D-21
D.16.2	Attributes	D-22
D.16.3	Example.....	D-22
D.17	caching-system	D-22
D.17.1	Child Elements	D-22
D.17.2	Attributes	D-23
D.17.3	Example.....	D-23
D.18	channel.....	D-23
D.18.1	Child Elements	D-23
D.18.2	Attributes	D-23
D.18.3	Example.....	D-23
D.19	channel (http-pub-sub-adapter Child Element)	D-24
D.19.1	Child Elements	D-24
D.19.2	Attributes	D-24
D.19.3	Example.....	D-24
D.20	coherence-cache-config	D-24
D.20.1	Child Elements	D-24
D.20.2	Attributes	D-24
D.20.3	Example.....	D-24
D.21	coherence-caching-system	D-25
D.21.1	Child Elements	D-25
D.21.2	Attributes	D-25
D.21.3	Example.....	D-25

D.22	coherence-cluster-config	D-25
D.22.1	Child Elements	D-25
D.22.2	Attributes	D-26
D.22.3	Example.....	D-26
D.23	collect-interval	D-26
D.23.1	Child Elements	D-26
D.23.2	Attributes	D-26
D.23.3	Example.....	D-26
D.24	concurrent-consumers.....	D-27
D.24.1	Child Elements	D-27
D.24.2	Attributes	D-27
D.24.3	Example.....	D-27
D.25	connection-jndi-name	D-27
D.25.1	Child Elements	D-28
D.25.2	Attributes	D-28
D.25.3	Example.....	D-28
D.26	connection-encrypted-password	D-28
D.26.1	Child Elements	D-28
D.26.2	Attributes	D-28
D.26.3	Example.....	D-28
D.27	connection-password.....	D-29
D.27.1	Child Elements	D-29
D.27.2	Attributes	D-29
D.27.3	Example.....	D-29
D.28	connection-user	D-29
D.28.1	Child Elements	D-30
D.28.2	Attributes	D-30
D.28.3	Example.....	D-30
D.29	database.....	D-30
D.29.1	Child Elements	D-30
D.29.2	Attributes	D-30
D.29.3	Example.....	D-30
D.30	dataset-name.....	D-31
D.30.1	Child Elements	D-31
D.30.2	Attributes	D-31
D.30.3	Example.....	D-31
D.31	delivery-mode	D-31
D.31.1	Child Elements	D-31
D.31.2	Attributes	D-31
D.31.3	Example.....	D-31
D.32	destination-jndi-name	D-32
D.32.1	Child Elements	D-32
D.32.2	Attributes	D-32
D.32.3	Example.....	D-32
D.33	destination-name.....	D-32
D.33.1	Child Elements	D-32
D.33.2	Attributes	D-32

D.33.3	Example.....	D-32
D.34	diagnostic-profiles	D-33
D.34.1	Child Elements	D-33
D.34.2	Attributes	D-33
D.34.3	Example.....	D-33
D.35	direction.....	D-33
D.35.1	Child Elements	D-33
D.35.2	Attributes	D-34
D.35.3	Example.....	D-34
D.36	duration	D-34
D.36.1	Child Elements	D-34
D.36.2	Attributes	D-34
D.36.3	Example.....	D-34
D.37	enabled	D-35
D.37.1	Child Elements	D-35
D.37.2	Attributes	D-35
D.37.3	Example.....	D-35
D.38	encrypted-password	D-36
D.38.1	Child Elements	D-36
D.38.2	Attributes	D-36
D.38.3	Example.....	D-36
D.39	end.....	D-36
D.39.1	Child Elements	D-37
D.39.2	Attributes	D-37
D.39.3	Example.....	D-37
D.40	end-location	D-37
D.40.1	Child Elements	D-37
D.40.2	Attributes	D-37
D.40.3	Example.....	D-38
D.41	event-bean	D-38
D.41.1	Child Elements	D-38
D.41.2	Attributes	D-38
D.41.3	Example.....	D-38
D.42	event-type.....	D-39
D.42.1	Child Elements	D-39
D.42.2	Attributes	D-39
D.42.3	Example.....	D-39
D.43	event-type-list.....	D-39
D.43.1	Child Elements	D-40
D.43.2	Attributes	D-40
D.43.3	Example.....	D-40
D.44	eviction-policy	D-40
D.44.1	Child Elements	D-40
D.44.2	Attributes	D-40
D.44.3	Example.....	D-40
D.45	group-binding	D-41
D.45.1	Child Elements	D-41

D.45.2	Attributes	D-41
D.45.3	Example.....	D-41
D.46	heartbeat.....	D-42
D.46.1	Child Elements	D-42
D.46.2	Attributes	D-42
D.46.3	Example.....	D-42
D.47	http-pub-sub-adapter	D-42
D.47.1	Child Elements	D-42
D.47.2	Attributes	D-43
D.47.3	Example.....	D-43
D.48	idle-time.....	D-43
D.48.1	Child Elements	D-43
D.48.2	Attributes	D-43
D.48.3	Example.....	D-44
D.49	jms-adapter	D-44
D.49.1	Child Elements	D-44
D.49.2	Attributes	D-45
D.49.3	Example.....	D-45
D.50	jndi-factory.....	D-45
D.50.1	Child Elements	D-45
D.50.2	Attributes	D-45
D.50.3	Example.....	D-45
D.51	jndi-provider-url	D-46
D.51.1	Child Elements	D-46
D.51.2	Attributes	D-46
D.51.3	Example.....	D-46
D.52	listeners.....	D-46
D.52.1	Child Elements	D-46
D.52.2	Attributes	D-46
D.52.3	Example.....	D-46
D.53	location	D-47
D.53.1	Child Elements	D-47
D.53.2	Attributes	D-47
D.53.3	Example.....	D-47
D.54	max-latency.....	D-48
D.54.1	Child Elements	D-48
D.54.2	Attributes	D-48
D.54.3	Example.....	D-48
D.55	max-size.....	D-49
D.55.1	Child Elements	D-49
D.55.2	Attributes	D-49
D.55.3	Example.....	D-49
D.56	max-threads	D-49
D.56.1	Child Elements	D-49
D.56.2	Attributes	D-50
D.56.3	Example.....	D-50
D.57	message-selector.....	D-50

D.57.1	Child Elements	D-50
D.57.2	Attributes	D-50
D.57.3	Example.....	D-50
D.58	name.....	D-50
D.58.1	Child Elements	D-51
D.58.2	Attributes	D-51
D.58.3	Example.....	D-51
D.59	netio.....	D-51
D.59.1	Child Elements	D-51
D.59.2	Attributes	D-51
D.59.3	Example.....	D-51
D.60	num-threads.....	D-51
D.60.1	Child Elements	D-52
D.60.2	Attributes	D-52
D.60.3	Example.....	D-52
D.61	param.....	D-52
D.61.1	Child Elements	D-52
D.61.2	Attributes	D-52
D.61.3	Example.....	D-52
D.62	parameter	D-53
D.62.1	Child Elements	D-53
D.62.2	Attributes	D-53
D.62.3	Example.....	D-53
D.63	params	D-53
D.63.1	Child Elements	D-54
D.63.2	Attributes	D-54
D.63.3	Example.....	D-54
D.64	password	D-54
D.64.1	Child Elements	D-55
D.64.2	Attributes	D-55
D.64.3	Example.....	D-55
D.65	playback-parameters	D-55
D.65.1	Child Elements	D-55
D.65.2	Attributes	D-56
D.65.3	Example.....	D-56
D.66	playback-speed.....	D-56
D.66.1	Child Elements	D-56
D.66.2	Attributes	D-56
D.66.3	Example.....	D-56
D.67	processor (EPL)	D-57
D.67.1	Child Elements	D-57
D.67.2	Attributes	D-57
D.67.3	Example.....	D-57
D.68	processor (Oracle CQL).....	D-58
D.68.1	Child Elements	D-58
D.68.2	Attributes	D-58
D.68.3	Example.....	D-58

D.69	profile.....	D-59
D.69.1	Child Elements	D-59
D.69.2	Attributes	D-59
D.69.3	Example.....	D-59
D.70	provider-name.....	D-60
D.70.1	Child Elements	D-60
D.70.2	Attributes	D-60
D.70.3	Example.....	D-61
D.71	query	D-61
D.71.1	Child Elements	D-61
D.71.2	Attributes	D-61
D.71.3	Example.....	D-61
D.72	record-parameters.....	D-62
D.72.1	Child Elements	D-62
D.72.2	Attributes	D-62
D.72.3	Example.....	D-62
D.73	repeat	D-62
D.73.1	Child Elements	D-63
D.73.2	Attributes	D-63
D.73.3	Example.....	D-63
D.74	rule	D-63
D.74.1	Child Elements	D-63
D.74.2	Attributes	D-63
D.74.3	Example.....	D-64
D.75	rules.....	D-64
D.75.1	Child Elements	D-64
D.75.2	Attributes	D-64
D.75.3	Example.....	D-64
D.76	schedule-time-range	D-65
D.76.1	Child Elements	D-65
D.76.2	Attributes	D-65
D.76.3	Example.....	D-65
D.77	schedule-time-range-offset	D-65
D.77.1	Child Elements	D-66
D.77.2	Attributes	D-66
D.77.3	Example.....	D-66
D.78	selector	D-66
D.78.1	Child Elements	D-67
D.78.2	Attributes	D-67
D.78.3	Example.....	D-67
D.79	server-context-path.....	D-68
D.79.1	Child Elements	D-68
D.79.2	Attributes	D-68
D.79.3	Example.....	D-68
D.80	server-url	D-68
D.80.1	Child Elements	D-69
D.80.2	Attributes	D-69

D.80.3	Example.....	D-69
D.81	session-ack-mode-name.....	D-69
D.81.1	Child Elements.....	D-69
D.81.2	Attributes.....	D-70
D.81.3	Example.....	D-70
D.82	session-transacted.....	D-70
D.82.1	Child Elements.....	D-70
D.82.2	Attributes.....	D-70
D.82.3	Example.....	D-70
D.83	stage.....	D-70
D.83.1	Child Elements.....	D-71
D.83.2	Attributes.....	D-71
D.83.3	Example.....	D-71
D.84	start.....	D-71
D.84.1	Child Elements.....	D-72
D.84.2	Attributes.....	D-72
D.84.3	Example.....	D-72
D.85	start-location.....	D-72
D.85.1	Child Elements.....	D-72
D.85.2	Attributes.....	D-72
D.85.3	Example.....	D-73
D.86	start-stage.....	D-73
D.86.1	Child Elements.....	D-73
D.86.2	Attributes.....	D-73
D.86.3	Example.....	D-73
D.87	store-policy-parameters.....	D-74
D.87.1	Child Elements.....	D-74
D.87.2	Attributes.....	D-74
D.87.3	Example.....	D-74
D.88	stream.....	D-74
D.88.1	Child Elements.....	D-74
D.88.2	Attributes.....	D-75
D.88.3	Example.....	D-75
D.89	symbol.....	D-75
D.89.1	Child Elements.....	D-75
D.89.2	Attributes.....	D-75
D.89.3	Example.....	D-75
D.90	symbols.....	D-75
D.90.1	Child Elements.....	D-76
D.90.2	Attributes.....	D-76
D.90.3	Example.....	D-76
D.91	threshold.....	D-76
D.91.1	Child Elements.....	D-76
D.91.2	Attributes.....	D-76
D.91.3	Example.....	D-76
D.92	throughput.....	D-77
D.92.1	Child Elements.....	D-77

D.92.2	Attributes	D-77
D.92.3	Example.....	D-77
D.93	throughput-interval.....	D-78
D.93.1	Child Elements	D-78
D.93.2	Attributes	D-78
D.93.3	Example.....	D-78
D.94	time-range	D-79
D.94.1	Child Elements	D-79
D.94.2	Attributes	D-79
D.94.3	Example.....	D-79
D.95	time-range-offset	D-79
D.95.1	Child Elements	D-79
D.95.2	Attributes	D-80
D.95.3	Example.....	D-80
D.96	time-to-live.....	D-80
D.96.1	Child Elements	D-80
D.96.2	Attributes	D-80
D.96.3	Example.....	D-80
D.97	unit	D-81
D.97.1	Child Elements	D-81
D.97.2	Attributes	D-81
D.97.3	Example.....	D-81
D.98	user.....	D-82
D.98.1	Child Elements	D-82
D.98.2	Attributes	D-82
D.98.3	Example.....	D-82
D.99	value.....	D-82
D.99.1	Child Elements	D-82
D.99.2	Attributes	D-82
D.99.3	Example.....	D-82
D.100	view.....	D-83
D.100.1	Child Elements	D-83
D.100.2	Attributes	D-83
D.100.3	Example.....	D-83
D.101	work-manager	D-84
D.101.1	Child Elements	D-84
D.101.2	Attributes	D-84
D.101.3	Example.....	D-84
D.102	work-manager-name	D-84
D.102.1	Child Elements	D-85
D.102.2	Attributes	D-85
D.102.3	Example.....	D-85
D.103	write-behind	D-85
D.103.1	Child Elements	D-85
D.103.2	Attributes	D-85
D.103.3	Example.....	D-85
D.104	write-none	D-86

D.104.1	Child Elements	D-86
D.104.2	Attributes	D-86
D.104.3	Example.....	D-86
D.105	write-through	D-86
D.105.1	Child Elements	D-87
D.105.2	Attributes	D-87
D.105.3	Example.....	D-87

E Schema Reference: Deployment deployment.xsd

E.1	Overview of the Oracle CEP Deployment Elements	E-1
E.1.1	Element Hierarchy	E-1
E.1.2	Example of an Oracle CEP Deployment Configuration File	E-1
E.2	wlevs:deployment.....	E-2
E.2.1	Child Elements	E-2
E.2.2	Attributes	E-2
E.2.3	Example.....	E-2

F Schema Reference: Server Configuration wlevs_server_config.xsd

F.1	Overview of the Oracle CEP Server Configuration Elements	F-1
F.1.1	Element Hierarchy	F-1
F.1.2	Example of an Oracle CEP Server Configuration File.....	F-2
F.2	auth-constraint	F-4
F.2.1	Child Elements	F-4
F.2.2	Attributes	F-4
F.2.3	Example.....	F-4
F.3	bdb-config	F-5
F.3.1	Child Elements	F-5
F.3.2	Attributes	F-5
F.3.3	Example.....	F-5
F.4	channels.....	F-6
F.4.1	Child Elements	F-6
F.4.2	Attributes	F-6
F.4.3	Example.....	F-6
F.5	channel-constraints	F-7
F.5.1	Child Elements	F-7
F.5.2	Attributes	F-7
F.5.3	Example.....	F-7
F.6	channel-resource-collection	F-7
F.6.1	Child Elements	F-8
F.6.2	Attributes	F-8
F.6.3	Example.....	F-8
F.7	cluster.....	F-9
F.7.1	Child Elements	F-9
F.7.2	Attributes	F-10
F.7.3	Example.....	F-10
F.8	connection-pool-params	F-10

F.8.1	Child Elements	F-10
F.8.2	Attributes	F-12
F.8.3	Example.....	F-12
F.9	cql	F-13
F.9.1	Child Elements	F-13
F.9.2	Attributes	F-13
F.9.3	Example.....	F-13
F.10	data-source.....	F-14
F.10.1	Child Elements	F-14
F.10.2	Attributes	F-14
F.10.3	Example.....	F-14
F.11	data-source-params	F-15
F.11.1	Child Elements	F-15
F.11.2	Attributes	F-16
F.11.3	Example.....	F-16
F.12	driver-params	F-17
F.12.1	Child Elements	F-17
F.12.2	Attributes	F-17
F.12.3	Example.....	F-17
F.13	domain.....	F-18
F.13.1	Child Elements	F-18
F.13.2	Attributes	F-18
F.13.3	Example.....	F-18
F.14	debug	F-18
F.14.1	Child Elements	F-18
F.14.2	Attributes	F-19
F.14.3	Example.....	F-19
F.15	event-store.....	F-19
F.15.1	Child Elements	F-19
F.15.2	Attributes	F-19
F.15.3	Example.....	F-19
F.16	exported-jndi-context	F-20
F.16.1	Child Elements	F-20
F.16.2	Attributes	F-20
F.16.3	Example.....	F-20
F.17	http-pubsub	F-20
F.17.1	Child Elements	F-20
F.17.2	Attributes	F-21
F.17.3	Example.....	F-21
F.18	jetty.....	F-21
F.18.1	Child Elements	F-21
F.18.2	Attributes	F-22
F.18.3	Example.....	F-22
F.19	jetty-web-app.....	F-22
F.19.1	Child Elements	F-22
F.19.2	Attributes	F-23
F.19.3	Example.....	F-23

F.20	jmx.....	F-23
F.20.1	Child Elements.....	F-23
F.20.2	Attributes.....	F-23
F.20.3	Example.....	F-24
F.21	jndi-context.....	F-24
F.21.1	Child Elements.....	F-24
F.21.2	Attributes.....	F-24
F.21.3	Example.....	F-24
F.22	log-file.....	F-24
F.22.1	Child Elements.....	F-25
F.22.2	Attributes.....	F-26
F.22.3	Example.....	F-26
F.23	log-stdout.....	F-26
F.23.1	Child Elements.....	F-26
F.23.2	Attributes.....	F-27
F.23.3	Example.....	F-27
F.24	logging-service.....	F-27
F.24.1	Child Elements.....	F-27
F.24.2	Attributes.....	F-27
F.24.3	Example.....	F-28
F.25	message-filters.....	F-28
F.25.1	Child Elements.....	F-28
F.25.2	Attributes.....	F-28
F.25.3	Example.....	F-28
F.26	name.....	F-29
F.26.1	Child Elements.....	F-29
F.26.2	Attributes.....	F-29
F.26.3	Example.....	F-29
F.27	netio.....	F-29
F.27.1	Child Elements.....	F-29
F.27.2	Attributes.....	F-30
F.27.3	Example.....	F-30
F.28	netio-client.....	F-30
F.28.1	Child Elements.....	F-30
F.28.2	Attributes.....	F-30
F.28.3	Example.....	F-30
F.29	path.....	F-30
F.29.1	Child Elements.....	F-31
F.29.2	Attributes.....	F-31
F.29.3	Example.....	F-31
F.30	pubsub-bean.....	F-31
F.30.1	Child Elements.....	F-31
F.30.2	Attributes.....	F-32
F.30.3	Example.....	F-32
F.31	rdbms-event-store-provider.....	F-32
F.31.1	Child Elements.....	F-32
F.31.2	Attributes.....	F-33

F.31.3	Example.....	F-33
F.32	rmi	F-33
F.32.1	Child Elements	F-33
F.32.2	Attributes	F-34
F.32.3	Example.....	F-34
F.33	scheduler	F-34
F.33.1	Child Elements	F-34
F.33.2	Attributes	F-35
F.33.3	Example.....	F-35
F.34	server-config	F-35
F.34.1	Child Elements	F-35
F.34.2	Attributes	F-36
F.34.3	Example.....	F-36
F.35	services	F-37
F.35.1	Child Elements	F-37
F.35.2	Attributes	F-37
F.35.3	Example.....	F-37
F.36	show-detail-error-message	F-38
F.36.1	Child Elements	F-38
F.36.2	Attributes	F-38
F.36.3	Example.....	F-38
F.37	ssl.....	F-39
F.37.1	Child Elements	F-39
F.37.2	Attributes	F-40
F.37.3	Example.....	F-40
F.38	timeout-seconds	F-40
F.38.1	Child Elements	F-40
F.38.2	Attributes	F-40
F.38.3	Example.....	F-40
F.39	transaction-manager	F-41
F.39.1	Child Elements	F-41
F.39.2	Attributes	F-43
F.39.3	Example.....	F-43
F.40	use-secure-connections.....	F-43
F.40.1	Child Elements	F-44
F.40.2	Attributes	F-44
F.40.3	Example.....	F-44
F.41	weblogic-instances	F-44
F.41.1	Child Elements	F-44
F.41.2	Attributes	F-44
F.41.3	Example.....	F-45
F.42	weblogic-jta-gateway.....	F-45
F.42.1	Child Elements	F-45
F.42.2	Attributes	F-45
F.42.3	Example.....	F-45
F.43	weblogic-rmi-client.....	F-46
F.43.1	Child Elements	F-46

F.43.2	Attributes	F-46
F.43.3	Example.....	F-46
F.44	work-manager	F-46
F.44.1	Child Elements	F-46
F.44.2	Attributes	F-47
F.44.3	Example.....	F-47
F.45	xa-params.....	F-47
F.45.1	Child Elements	F-47
F.45.2	Attributes	F-49
F.45.3	Example.....	F-49

G Schema Reference: Message Catalog msgcat.dtd

G.1	Overview of the Message Catalog Elements.....	G-1
G.1.1	Element Hierarchy	G-1
G.1.2	Examples	G-1
G.2	message_catalog.....	G-2
G.2.1	Child Elements	G-2
G.2.2	Attributes	G-3
G.2.3	Example.....	G-4
G.3	logmessage	G-5
G.3.1	Child Elements	G-5
G.3.2	Attributes	G-5
G.3.3	Example.....	G-6
G.4	message	G-7
G.4.1	Child Elements	G-7
G.4.2	Attributes	G-7
G.4.3	Example.....	G-8
G.5	messagebody	G-8
G.5.1	Child Elements	G-9
G.5.2	Attributes	G-9
G.5.3	Example.....	G-9
G.6	messagedetail	G-10
G.6.1	Child Elements	G-10
G.6.2	Attributes	G-10
G.6.3	Example.....	G-10
G.7	cause.....	G-10
G.7.1	Child Elements	G-11
G.7.2	Attributes	G-11
G.7.3	Example.....	G-11
G.8	action.....	G-11
G.8.1	Child Elements	G-11
G.8.2	Attributes	G-11
G.8.3	Example.....	G-12

H Schema Reference: Locale Message Catalog l10n_msgcat.dtd

H.1	Overview of the Locale Message Catalog Elements	H-1
-----	---	-----

H.1.1	Element Hierarchy	H-1
H.1.2	Examples	H-1
H.2	locale_message_catalog.....	H-2
H.2.1	Child Elements	H-2
H.2.2	Attributes	H-2
H.2.3	Example.....	H-3
H.3	logmessage	H-3
H.3.1	Child Elements	H-3
H.3.2	Attributes	H-4
H.3.3	Example.....	H-4
H.4	message	H-5
H.4.1	Child Elements	H-5
H.4.2	Attributes	H-5
H.4.3	Example.....	H-5
H.5	messagebody	H-5
H.5.1	Child Elements	H-5
H.5.2	Attributes	H-5
H.5.3	Example.....	H-6
H.6	messagedetail.....	H-6
H.6.1	Child Elements	H-6
H.6.2	Attributes	H-6
H.6.3	Example.....	H-6
H.7	cause.....	H-7
H.7.1	Child Elements	H-7
H.7.2	Attributes	H-7
H.7.3	Example.....	H-7
H.8	action.....	H-8
H.8.1	Child Elements	H-8
H.8.2	Attributes	H-8
H.8.3	Example.....	H-8

I Oracle CEP Metadata Annotation Reference

I.1	Overview of Oracle CEP Metadata Annotations	I-1
I.1.1	Adapter Lifecycle Annotations	I-1
I.1.2	OSGi Service Reference Annotations	I-1
I.1.3	Resource Access Annotations	I-2
I.2	com.bea.wlevs.configuration.Activate.....	I-2
I.2.1	Example.....	I-2
I.3	com.bea.wlevs.configuration.Prepare.....	I-4
I.3.1	Example.....	I-4
I.4	com.bea.wlevs.configuration.Rollback	I-5
I.4.1	Example.....	I-5
I.5	com.bea.wlevs.util.Service.....	I-6
I.5.1	Attributes	I-6
I.5.2	Example.....	I-7

J Oracle CEP IDE for Eclipse Tutorial

J.1	Before You Begin.....	J-1
J.2	Step 1: Create an Oracle CEP Definition.....	J-2
J.3	Step 2: Create an Oracle CEP Application.....	J-3
J.4	Step 3: Start the Oracle CEP Server and Deploy the Project.....	J-5
J.5	Step 4: Change Code and Redeploy	J-6
J.6	Step 5: Debug the Deployed Application.....	J-8
J.7	Next Steps	J-9

Index

List of Examples

1-1	EPN Assembly File With Nested Bean	1-4
1-2	EPN Assembly File With all Nodes Nested	1-5
1-3	Application 1 Referencing Foreign Stage in Application 2.....	1-5
1-4	Foreign Stage in Application 2.....	1-6
1-5	netio Element	1-13
1-6	Accessing the netio Element port Value	1-13
1-7	Adding a ConfigurationPropertyPlaceholderConfigurer.....	1-13
1-8	Sample Resource: Data Source StockDS	1-22
1-9	Static Resource Injection Using Static Resource Names: Annotations.....	1-23
1-10	Static Resource Injection Using Static Resource Names: XML.....	1-23
1-11	Static Resource Injection Using Static Resource Names: Annotations.....	1-23
1-12	Static Resource Injection Using Static Resource Names: XML.....	1-23
1-13	Custom Component Configuration	1-23
1-14	Static Resource Injection Using Dynamic Resource Names: Annotations	1-24
1-15	Static Resource Injection Using Dynamic Resource Names: XML	1-24
1-16	Dynamic Resource Injection: Annotations	1-24
1-17	Dynamic Resource Lookup Using JNDI.....	1-25
2-1	Event Type Repository	2-3
2-2	Specifying Java Data Types for java.util.Map Event Type Properties	2-4
2-3	Specifying com.bea.welvs.ede.api.Type Data Types for Tuple Event Type Properties ...	2-4
2-4	MarketEvent Class	2-7
2-5	MarketEvent Class	2-10
2-6	EPN Assembly File event-type-repository	2-11
2-7	Programmatically Registering an Event.....	2-11
2-8	EPN Assembly File event-type-repository	2-16
2-9	ForeignExchangeEvent Class: Does not Conform to JavaBean Standards.....	2-17
2-10	ForeignExchangeBuilderFactory	2-18
2-11	EPN Assembly File event-type-repository	2-19
2-12	Programmatically Registering an Event.....	2-19
2-13	EPN Assembly File event-type-repository	2-20
2-14	Programmatically Registering an Event.....	2-20
2-15	EPN Assembly File With OSGi Reference to EventTypeRepository.....	2-22
2-16	Accessing the EventTypeRepository in the MyBean Implementation	2-23
2-17	Java Source File Using the @ServiceReference Annotation	2-23
2-18	Java Source File Using the @Service Annotation	2-23
3-1	Default eclipse.ini File	3-11
3-2	Memory Resources	3-11
3-3	Virtual Machine Path.....	3-12
4-1	Accessing a Properties File	4-38
6-1	Assembly Source for EPN With Nested Bean.....	6-9
6-2	Assembly Source for EPN With all Nodes Nested	6-10
7-1	wlvs:adapter Element for Inbound Adapter.....	7-5
7-2	jms-adapter Element for Inbound Adapter.....	7-5
7-3	jms-adapter Element With Tibco EMS JMS Configuration.....	7-6
7-4	wlvs:adapter Element for Inbound Adapter.....	7-7
7-5	jms-adapter Element for Inbound Adapter.....	7-8
7-6	jms-adapter Elements for an Oracle WebLogic Server JMS Provider	7-8
7-7	wlvs:adapter Element for Inbound Adapter.....	7-10
7-8	jms-adapter Element for Inbound Adapter.....	7-10
7-9	jms-adapter Element With Tibco EMS JMS Configuration.....	7-11
7-10	Custom Converter for an Inbound JMS Adapter	7-13
7-11	Specifying a Converter Class for an Inbound JMS Adapter in the EPN Assembly File	7-13
7-12	Custom Converter for an Outbound JMS Adapter.....	7-14
7-13	Specifying a Converter Class for an Outbound JMS Adapter in the EPN Assembly File	

	7-15	
9-1	EPN Assembly File Channel Id: priceStream	9-1
9-2	Component Configuration File Channel Name: priceStream	9-1
9-3	Channel as Relation: primary-key Attribute.....	9-3
9-4	PriceEvent	9-4
9-5	filterFanoutProcessor Oracle CQL Queries.....	9-5
9-6	Using selector to Control Which Query Results are Output	9-5
9-7	Batch Processing Channel.....	9-6
9-8	Component Configuration File Header and config Element	9-8
9-9	Component Configuration File Channel Element	9-8
9-10	EPN Assembly File Channel Id: priceStream	9-8
9-11	Component Configuration File Channel Name: priceStream	9-8
9-12	Component Configuration File Header and config Element	9-11
9-13	Component Configuration File Channel Element	9-11
9-14	EPN Assembly File Channel Id: priceStream	9-12
9-15	Component Configuration File Channel Name: priceStream	9-12
9-16	Sample Channel Component Configuration File.....	9-16
9-17	Channel EPN Assembly File	9-17
10-1	EPN Assembly File Oracle CQL Processor Id: proc.....	10-2
10-2	Component Configuration File Oracle CQL Processor Name: proc	10-2
10-3	Default Processor Component Configuration	10-4
10-4	Table Create SQL Statement.....	10-7
10-5	Oracle CQL Query on Relational Database Table Stock	10-7
10-6	Oracle CEP Server config.xml File With Data Source StockDS.....	10-7
10-7	EPN Assembly File table Element	10-8
10-8	EPN Assembly File table-source Element	10-9
10-9	EPN Assembly File event-type element for a Table	10-9
10-10	Oracle CQL Query Using Table Event Type StockEvent.....	10-11
11-1	EPN Assembly File EPL Processor Id: proc	11-2
11-2	Component Configuration File EPL Processor Name: proc	11-2
12-1	EPN Assembly File Caching System Id: cacheSystem	12-2
12-2	Component Configuration File Caching System Name: cacheSystem	12-2
12-3	EPN Assembly File Caching Id: cache1	12-3
12-4	Component Configuration File Caching Name: cache1.....	12-3
12-5	Application Configuration File: Coherence Cache	12-17
12-6	Oracle Coherence Cache LocalListener Implementation.....	12-22
12-7	Oracle Coherence Cache EPN Assembly File for a Cache Loader.....	12-23
12-8	Oracle Coherence Cache LocalLoader Implementation.....	12-23
12-9	Oracle Coherence Cache EPN Assembly File for a Cache Store	12-24
12-10	Oracle Coherence Cache LocalStore Implementation	12-24
12-11	Valid Oracle CQL Query Against a Cache.....	12-29
13-1	bdb-config Element.....	13-5
13-2	Default bdb-config Element	13-11
17-1	bea-jaxws.xml File.....	17-3
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	Annotated Custom Adapter Implementation	19-3
19-2	Extended Component Configuration: Annotations.....	19-4
19-3	Extended Component Configuration File: XSD	19-6
19-4	Custom Adapter Implementation	19-9
19-5	Extended Component Configuration.....	19-10
21-1	Simple Failover EPN Assembly File	21-2
21-2	Simple Failover Component Configuration Assembly File.....	21-2
21-3	Simple Failover EPN Assembly File: Buffering Output Adapter	21-3
21-4	Application Timestamp Configuration	21-4
21-5	Configuring windowLength in the Buffering Output Adapter.....	21-4
21-6	Simple Failover Component Configuration File With High Availability Adapters.....	21-4
21-7	Simple Failover With Buffering EPN Assembly File	21-5
21-8	Simple Failover With Buffering Component Configuration Assembly File	21-6
21-9	Simple Failover EPN Assembly File: Buffering Output Adapter	21-7
21-10	Application Timestamp Configuration	21-7
21-11	Configuring windowLength in the Buffering Output Adapter.....	21-8
21-12	Simple Failover With Buffering Component Configuration File.....	21-8
21-13	Light-Weight Queue Trimming EPN Assembly File.....	21-9
21-14	Light-Weight Queue Trimming Component Configuration Assembly File.....	21-9
21-15	Light-Weight Queue Trimming EPN Assembly File: High Availability Input Adapter	21-10
21-16	Light-Weight Queue Trimming EPN Assembly File: Broadcast Output Adapter.....	21-11
21-17	High Availability Input Adapter: Default Configuration.....	21-12
21-18	High Availability Input Adapter: Tuple Events.....	21-12
21-19	High Availability Input Adapter: Key of One Event Property	21-12
21-20	High Availability Input Adapter: Key of Multiple Event Properties.....	21-13
21-21	MyCompoundKeyClass Implementation	21-13
21-22	Application Timestamp Configuration	21-14
21-23	Broadcast Output Adapter: Default Configuration	21-14
21-24	Broadcast Output Adapter: Key of One Event Property	21-14
21-25	Broadcast Output Adapter: Key of Multiple Event Properties	21-14
21-26	MyCompoundKeyClass Implementation	21-15
21-27	Light-Weight Queue Trimming Component Configuration File	21-15
21-28	Precise Recovery With JMS EPN Assembly File	21-17
21-29	Precise Recovery With JMS Component Configuration Assembly File	21-18
21-30	Precise Recovery With JMS EPN Assembly File: High Availability Input Adapter....	21-18
21-31	Precise Recovery With JMS EPN Assembly File: Correlating Output Adapter	21-19
21-32	High Availability Input Adapter: Default Configuration.....	21-20
21-33	High Availability Input Adapter: Tuple Events.....	21-20
21-34	High Availability Input Adapter: Key of One Event Property	21-20
21-35	High Availability Input Adapter: Key of Multiple Event Properties.....	21-21
21-36	MyCompoundKeyClass Implementation	21-21
21-37	Application Timestamp Configuration	21-22
21-38	Correlating Output Adapter Configuration: failOverDelay	21-22
21-39	Inbound JMS Adapter Assembly File	21-22
21-40	Inbound JMS Adapter Component Configuration File.....	21-22
21-41	Creating the Correlated Source.....	21-23
21-42	Correlating Output Adapter: correlatedSource.....	21-23
21-43	Inbound and Outbound JMS Adapter Component Configuration File	21-24
21-44	High Availability Input and Output Adapter Component Configuration File.....	21-24
21-45	High Availability Input Adapter EPN Assembly File.....	21-26
21-46	High Availability Input Adapter Component Configuration File	21-27
21-47	Buffering Output Adapter EPN Assembly File.....	21-28
21-48	Buffering Output Adapter Component Configuration File	21-29

21-49	Broadcast Output Adapter EPN Assembly File	21-29
21-50	Broadcast Output Adapter Component Configuration File.....	21-30
21-51	Correlating Output Adapter EPN Assembly File	21-31
21-52	Correlating Output Adapter Component Configuration File	21-32
22-1	ActiveActiveGroupBean bean Element	22-4
22-2	Common jms-adapter Selector Definitions	22-5
23-1	Definition of Event Type PriceEvent.....	23-1
23-2	EventPartitioner Class.....	23-5
23-3	EventPartitioner Class Implementation	23-6
23-4	ActiveActiveGroupBean bean Element	23-10
23-5	jms-adapter Selector Definition for ocep-server-1	23-10
23-6	Precise Recovery With JMS EPN Assembly File	23-11
23-7	Precise Recovery With JMS Component Configuration Assembly File	23-12
23-8	ActiveActiveGroupBean bean Element	23-15
23-9	jms-adapter Element for Inbound JMS Adapters.....	23-15
23-10	jms-adapter Selector Definition for ocep-server-1	23-15
23-11	jms-adapter Element for Outbound JMS Adapters.....	23-16
23-12	ActiveActiveGroupBean bean Element With groupPattern Attribute	23-17
24-1	bundler.sh Command Line Options	24-19
24-2	Using the Bundler Utility.....	24-20
24-3	Bundle JAR Contents.....	24-20
24-4	Service Registration Log Messages	24-20
24-5	MyActivator Class Implementation	24-23
24-6	Un-JAR the Database Driver	24-28
24-7	Adding Export-Package to the Manifest Editor	24-28
24-8	Adding a Bundle-Activator Element to the Manifest Editor.....	24-29
24-9	Adding a DynamicImport-Package Element to the Manifest Editor	24-29
24-10	Message Arguments	24-34
24-11	Log Message Catalog.....	24-35
24-12	Simple Text Catalog.....	24-35
24-13	Locale-Specific Catalog	24-36
25-1	EmployeeEvent Event Type	25-3
25-2	Data Feed File for EmployeeEvent Event Type.....	25-3
26-1	Event Inspector JSON Event.....	26-3
26-2	Event Inspector Service Local HTTP Pub-Sub Server	26-5
26-3	Oracle CEP Built-In HTTP Pub-Sub Server http-pubsub Element.....	26-5
26-4	Event Inspector Service Remote HTTP Pub-Sub Server	26-6
26-5	Oracle CEP Built-In HTTP Pub-Sub Server http-pubsub Element.....	26-6
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-20
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-22
C-9	Invoking the Aggregate User-Defined Function on an Oracle CQL Processor	C-22
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-24
C-14	User Defined Function Using Reference	C-24
D-1	adapter Element Hierarchy	D-1
D-2	http-pub-sub-adapter Element Hierarchy.....	D-2
D-3	jms-adapter Element Hierarchy	D-3

D-4	processor (EPL) Element Hierarchy	D-5
D-5	processor (Oracle CQL) Element Hierarchy	D-6
D-6	stream Element Hierarchy	D-6
D-7	channel Element Hierarchy	D-7
D-8	event-bean Element Hierarchy	D-8
D-9	caching-system Element Hierarchy	D-9
D-10	coherence-caching-system Element Hierarchy	D-9
D-11	diagnostic-profiles Element Hierarchy	D-9
D-12	filterFanoutProcessor Oracle CQL Queries	D-66
D-13	Using selector to Control Which Query Results are Output	D-67
G-1	Log Message Catalog Hierarchy	G-1
G-2	Simple Text Catalog Hierarchy	G-1
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-1
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-6
I-5	@Service Annotation	I-7

List of Figures

1-1	The HelloWorld Example Event Processing Network	1-9
1-2	Oracle CEP Application Lifecycle State Diagram	1-15
2-1	EPN Editor	2-8
2-2	Event Type Repository Editor	2-8
2-3	Event Type Repository Editor - JavaBean Event	2-9
2-4	EPN Editor	2-13
2-5	Event Type Repository Editor	2-13
2-6	Event Type Repository Editor - Tuple Event	2-14
3-1	Install Dialog	3-3
3-2	Add Site Dialog	3-3
3-3	Install Dialog - Site Selected	3-4
3-4	Install Dialog - Install Details	3-5
3-5	About Eclipse	3-5
3-6	About Eclipse Features Dialog	3-6
3-7	Feature Plug-ins Dialog	3-6
3-8	Install Dialog	3-8
3-9	Add Site Dialog	3-8
3-10	Select Local Site Archive Dialog	3-9
3-11	About Eclipse	3-9
3-12	About Eclipse Features Dialog	3-10
3-13	Feature Plug-ins Dialog	3-10
3-14	Configuration Details for Java 6	3-12
4-1	Oracle CEP Project Structure	4-2
4-2	New Project - Select a Wizard Dialog	4-3
4-3	New Oracle CEP Application Project Wizard: Create an Oracle CEP Application	4-4
4-4	New Oracle CEP Application Project Wizard: Oracle CEP Application Content	4-5
4-5	New Oracle CEP Application Project Wizard: Template Dialog	4-6
4-6	New Dialog	4-7
4-7	New CEP Assembly File Dialog	4-8
4-8	New Dialog	4-9
4-9	New CEP Application Configuration File Dialog	4-10
4-10	Oracle CEP Project build.properties File	4-11
4-11	Export Dialog	4-12
4-12	Oracle CEP Applications Export: Select Project Dialog	4-12
4-13	Workspace Launcher Dialog	4-14
4-14	Import Dialog	4-15
4-15	Import Projects Dialog	4-16
4-16	Project Properties Dialog: Project Facets	4-17
4-17	Modify Faceted Project	4-17
4-18	Preferences Dialog	4-18
4-19	Project Properties Dialog: Targeted Runtimes	4-19
4-20	Builder Error	4-20
4-21	Workspace Launcher Dialog	4-21
4-22	Import Dialog	4-22
4-23	Import Projects Dialog	4-23
4-24	Project Properties Dialog: Project Facets	4-24
4-25	Modify Faceted Project	4-24
4-26	Preferences Dialog	4-25
4-27	Project Properties Dialog: Targeted Runtimes	4-26
4-28	Builder Error	4-27
4-29	Preferences Dialog	4-28
4-30	Oracle CEP IDE for Eclipse lib Directory	4-30
4-31	Manifest Editor: Build Tab	4-31
4-32	Manifest Editor - Runtime Tab	4-32

4-33	JAR Selection Dialog.....	4-32
4-34	Manifest Editor Runtime tab After Adding a JAR to the Classpath	4-33
4-35	Manifest Editor MANIFEST.MF Tab	4-34
4-36	Package Explorer.....	4-34
4-37	Manifest Editor: Dependencies Tab	4-35
4-38	Plug-in Selection Dialog.....	4-36
4-39	Oracle CEP IDE for Eclipse properties Directory.....	4-37
4-40	Manifest Editor: Build Tab.....	4-37
4-41	Oracle CEP IDE for Eclipse lib Directory	4-38
4-42	Manifest Editor: Runtime tab.....	4-39
4-43	Package Selection Dialog	4-39
4-44	Manifest Editor Runtime tab After Exporting a Package	4-40
4-45	Oracle CEP IDE for Eclipse lib Directory	4-41
4-46	Manifest Editor: Dependencies tab	4-41
4-47	Package Selection Dialog	4-42
4-48	Manifest Editor Dependencies tab After Importing a Package.....	4-42
4-49	Properties Dialog	4-44
4-50	Oracle CEP Problem Severities Dialog: Workspace.....	4-45
4-51	Oracle CEP Problem Severities Dialog: Project	4-46
5-1	Oracle CEP IDE for Eclipse Server View	5-4
5-2	New Server: Define New Server Dialog (No Installed Runtimes)	5-4
5-3	New Server: New Oracle CEP v11.1 Runtime Dialog	5-5
5-4	New Server: Define New Server (Installed Runtimes) Dialog.....	5-6
5-5	New Server: New Oracle CEP v11.1 Server.....	5-7
5-6	New Server: New Oracle CEP v11 Server Dialog for a Local Server	5-8
5-7	Preferences - Server	5-9
5-8	Oracle CEP IDE for Eclipse Server View	5-10
5-9	New Server: Define New Server Dialog (No Installed Runtimes)	5-11
5-10	New Server: New Oracle CEP v11.1 Runtime Dialog	5-12
5-11	New Server: Define New Server (Installed Runtimes) Dialog.....	5-13
5-12	New Server: New Oracle CEP v11.1 Server.....	5-14
5-13	New Server: New Oracle CEP v11 Server Dialog for a Remote Server	5-15
5-14	Preferences - Server - Installed Runtimes.....	5-16
5-15	New Server Runtime Environment Dialog	5-17
5-16	New Server Runtime Environment: New Oracle CEP v11.1 Runtime Dialog.....	5-18
5-17	Starting an Oracle CEP Server.....	5-20
5-18	Stopping an Oracle CEP Server	5-20
5-19	Attaching to an Existing Local Oracle CEP Server Instance.....	5-21
5-20	Attach to Running CEP Server.....	5-21
5-21	Attaching to an Existing Remote Oracle CEP Server Instance.....	5-22
5-22	Stopping an Oracle CEP Server	5-23
5-23	Adding a Project to an Oracle CEP Server	5-24
5-24	Add and Remove Dialog	5-24
5-25	Server View After Adding a Project.....	5-25
5-26	Select Cluster Deployment Group Name Dialog	5-25
5-27	Server View After Deploying (Publishing) a Project.....	5-26
5-28	Server Overview Editor	5-26
5-29	Editing the Domain Configuration File	5-30
5-30	Oracle CEP Domain Configuration File config.xml.....	5-30
5-31	Opening the Oracle CEP Visualizer	5-31
5-32	Oracle CEP Visualizer	5-32
5-33	Setting a Breakpoint.....	5-33
5-34	Starting the Oracle CEP Server in Debug Mode.....	5-33
6-1	Opening the EPN Editor from a Project	6-2
6-2	EPN Editor	6-2

6-3	Opening the EPN Editor from a Context or Configuration File	6-3
6-4	EPN Editor	6-4
6-5	EPN Flow Representation.....	6-5
6-6	Filtering the EPN by Assembly File	6-5
6-7	Zoom Level	6-6
6-8	Optimize Layout	6-6
6-9	Show/Hide Unconnected Beans	6-7
6-10	Exporting the EPN as an Image File.....	6-7
6-11	Printing the EPN	6-7
6-12	Configuration Badging.....	6-8
6-13	Link Source	6-8
6-14	Link Source Assembly File	6-8
6-15	Link Listener	6-8
6-16	Link Listener Assembly File	6-9
6-17	EPN With Nested Bean	6-9
6-18	EPN With all Nodes Nested	6-10
6-19	Event Type Repository Editor.....	6-10
6-20	Node with Configuration Badge	6-11
6-21	Component Configuration File: Hyperlinking to EPN Assembly File	6-12
6-22	EPN Assembly File: Hyperlinking to Component Configuration File	6-13
6-23	Oracle CQL Statement: Event Schema.....	6-13
6-24	Corresponding Event Definition in EPN Assembly File.....	6-14
6-25	Example Oracle CEP EPN	6-15
6-26	Oracle CEP Type Browser	6-16
6-27	Opening the FilterAsia EPN Assembly File.....	6-17
6-28	Opening the FilterAsia Component Configuration File.....	6-17
6-29	Creating a Basic Node	6-20
6-30	New Basic Node.....	6-20
6-31	Creating an Adapter Node	6-21
6-32	New Adapter Wizard	6-21
6-33	New Adapter Wizard - jms-inbound	6-23
6-34	New Adapter Wizard - jms-outbound.....	6-24
6-35	New Adapter Wizard - httppub	6-24
6-36	New Adapter Wizard - httpsub	6-25
6-37	New Adapter Node	6-25
6-38	Creating a Processor Node	6-26
6-39	New Processor Dialog.....	6-27
6-40	New Processor Node.....	6-27
6-41	Connecting Nodes: Connection Allowed.....	6-28
6-42	Connecting Nodes: Connection Forbidden.....	6-29
6-43	Valid Connections.....	6-29
6-44	EPN Assembly File: Before Connection.....	6-29
6-45	EPN Assembly File: After Connection.....	6-30
6-46	Laying Out Nodes.....	6-30
6-47	Renaming Nodes.....	6-30
6-48	Deleting Nodes.....	6-31
6-49	EPN Before Deleting a Channel Node	6-31
6-50	Assembly File Before Deleting a Channel Node	6-31
6-51	EPN After Deleting a Channel Node	6-32
6-52	Assembly File After Deleting a Channel Node	6-32
8-1	Built-In Pub-Sub Adapter For Local Publishing	8-3
8-2	Built-In Pub-Sub Adapter For Remote Publishing.....	8-3
8-3	Built-In Pub-Sub Adapter For Subscribing	8-4
9-1	EPN With Oracle CQL Processor and Down-Stream Channel	9-5
9-2	Channel With Configuration Badge.....	9-10

9-3	Channel With Configuration Badge.....	9-13
9-4	EPN with Two Channels.....	9-16
12-1	Editing the MANIFEST.MF File.....	12-11
12-2	Editing the MANIFEST.MF File.....	12-28
12-3	Cache as Processor Source.....	12-31
12-4	Cache as Processor Sink.....	12-31
13-1	Configuring Record and Playback in an EPN.....	13-1
20-1	Oracle CEP High Availability: Primary and Secondary Servers.....	20-2
20-2	Oracle CEP High Availability Lifecycle State Diagram.....	20-2
20-3	Secondary Failure.....	20-3
20-4	Primary Failure and Failover.....	20-3
20-5	High Availability Adapters in the EPN.....	20-5
20-6	High Availability and Scalability.....	20-8
20-7	Precise Recovery with JMS.....	20-11
20-8	Event Order.....	20-16
21-1	Simple Failover EPN.....	21-2
21-2	Simple Failover With Buffering EPN.....	21-5
21-3	Light-Weight Queue Trimming EPN.....	21-9
21-4	Precise Recovery With JMS EPN.....	21-17
22-1	Event Partitioner EPN.....	22-2
22-2	Oracle CEP ActiveActiveGroupBean Without High Availability.....	22-5
22-3	Oracle CEP ActiveActiveGroupBean With High Availability.....	22-6
23-1	EventPartitioner EPN.....	23-1
23-2	New Java Class Dialog.....	23-5
23-3	Precise Recovery With JMS EPN.....	23-11
23-4	Oracle CEP ActiveActiveGroupBean With High Availability.....	23-13
24-1	Foreign Stage Dependency Graph.....	24-11
24-2	Preferences Dialog: Application Library Path.....	24-15
24-3	Select Path Variable Dialog.....	24-16
24-4	New Variable Dialog.....	24-16
24-5	Select Path Variable: With Variable.....	24-17
24-6	Variable Extension Dialog.....	24-17
24-7	Preferences Dialog: Application Library Path With Path Variable.....	24-18
24-8	Oracle CEP IDE for Eclipse lib Directory.....	24-21
24-9	New Java Class Dialog.....	24-22
24-10	Manifest Editor: Overview Tab.....	24-24
24-11	Manifest Editor: Runtime Tab.....	24-25
24-12	JAR Selection Dialog.....	24-25
24-13	Manifest Editor: Dependencies Tab.....	24-26
24-14	Package Selection Dialog.....	24-27
24-15	Manifest Editor.....	24-28
D-1	EPN With Oracle CQL Processor and Down-Stream Channel.....	D-66
J-1	Perspective Shortcut Menu.....	J-1
J-2	Oracle CEP IDE for Eclipse.....	J-2
J-3	Configuring the Oracle CEP Server for Automatic Publishing.....	J-3
J-4	Oracle CEP IDE for Eclipse Server View.....	J-3
J-5	New Oracle CEP Application Project Dialog.....	J-4
J-6	Templates Dialog.....	J-4
J-7	Project Explorer.....	J-5
J-8	Starting the Oracle CEP Server.....	J-5
J-9	Add and Remove Projects Dialog.....	J-6
J-10	Editing Java Source for an Adapter.....	J-7
J-11	Java Editor for helloworldAdapter.....	J-7
J-12	Manually Publishing a Project.....	J-8
J-13	Java Editor for HelloWorldBean.....	J-8

J-14	HelloWorldBean in the Debugger	J-9
------	--------------------------------------	-----

List of Tables

1-1	Resource Name Resolution.....	1-25
2-1	csvgen Adapter Types.....	2-5
3-1	New Update Site Dialog Attributes	3-4
3-2	Oracle CEP IDE for Eclipse Plug-Ins.....	3-7
3-3	Oracle CEP IDE for Eclipse Plug-Ins.....	3-11
4-1	Oracle CEP Project Artifacts.....	4-2
4-2	Create an Oracle CEP Application Dialog	4-4
4-3	Oracle CEP Application Content Dialog	4-5
4-4	New CEP Assembly File Dialog	4-8
4-5	New CEP Configuration File Dialog.....	4-10
4-6	Oracle CEP Application Content Dialog	4-13
4-7	Oracle CEP Problem Severities	4-47
5-1	Eclipse and Oracle CEP Server Concepts	5-1
5-2	New Server: Define New Server Dialog (No Installed Runtimes) Attributes	5-5
5-3	New Server: New Oracle CEP v11 Runtime Dialog Attributes	5-6
5-4	New Server: Define New Server (Installed Runtimes) Dialog Attributes	5-7
5-5	New Server: New Oracle CEP v11 Server Dialog Attributes for a Local Server	5-8
5-6	New Server: Define New Server Dialog (No Installed Runtimes) Attributes	5-11
5-7	New Server: New Oracle CEP v11 Runtime Dialog Attributes	5-12
5-8	New Server: Define New Server (Installed Runtimes) Dialog Attributes	5-13
5-9	New Server: New Oracle CEP v11 Server Dialog Attributes for a Local Server	5-15
5-10	New Server Runtime Dialog Attributes	5-17
5-11	New Server Runtime Dialog Attributes	5-18
5-12	Add and Remove Dialog Attributes	5-24
5-13	Server Overview Editor Attributes	5-27
6-1	Oracle CEP Type Dialog	6-16
6-2	EPN Editor Icons.....	6-18
6-3	New Adapter Wizard - Page 1	6-22
6-4	New Processor Dialog	6-27
7-1	jms-adapter Inbound Child Elements.....	7-19
7-2	jms-adapter Outbound Component Configuration Child Elements.....	7-22
8-1	http-pub-sub-adapter for Publishing Component Configuration Child Elements.....	8-13
8-2	http-pub-sub-adapter for Subscribing Component Configuration Child Elements.....	8-15
10-1	EPN Assembly File table Element Attributes	10-9
10-2	EPN Assembly File event-type Element Property Attributes	10-9
10-3	SQL Column Types and Oracle CEP Type Equivalents.....	10-10
13-1	Child Elements of bdb-config.....	13-5
13-2	Child Elements of record-parameters	13-6
13-3	Child Elements of playback-parameters	13-9
17-1	bea-jaxws.xml File Attributes.....	17-3
18-1	spatial:context Element Attributes	18-2
20-1	Oracle CEP High Availability Quality of Service.....	20-9
20-2	Oracle CEP High Availability Application Types.....	20-14
21-1	Child Elements of wlevs:adapter for the High Availability Input Adapter.....	21-26
21-2	High Availability Input Adapter Instance Properties	21-26
21-3	Child Elements of ha-inbound-adapter for the High Availability Input Adapter.....	21-27
21-4	Child Elements of wlevs:adapter for the Buffering Output Adapter.....	21-28
21-5	Buffering Output Adapter Instance Properties	21-28
21-6	Child Elements of ha-buffering-adapter for the Buffering Output Adapter	21-29
21-7	Child Elements of wlevs:adapter for the Broadcast Output Adapter.....	21-30
21-8	Broadcast Output Adapter Instance Properties	21-30
21-9	Child Elements of ha-broadcast-adapter for the Broadcast Output Adapter	21-31
21-10	Child Elements of wlevs:adapter for the Correlating Output Adapter	21-32

21-11	Correlating Output Adapter Instance Properties.....	21-32
21-12	Child Elements of ha-correlating-adapter for the Correlating Output Adapter	21-32
22-1	Event Partitioner Channel Threading Options.....	22-3
23-1	New Java Class Options for EventPartitioner	23-5
23-2	Oracle CEP Server Configuration File groups Element Configuration	23-9
23-3	Oracle CEP Server Configuration File groups Element Configuration	23-14
24-1	Oracle CEP Application Library Path.....	24-15
24-2	Oracle CEP Application Library Path Variable.....	24-16
24-3	bundler.sh Command Line Options	24-19
24-4	Factory Class and Service Interface	24-20
24-5	New Java Class Parameters	24-22
24-6	weblogic.i18ngen Utility Options.....	24-37
25-1	Load Generator Properties	25-3
26-1	Event Inspector JSON Event Required Attributes	26-3
C-1	Attributes of the wlevs:adapter Application Assembly Element	C-3
C-2	Attributes of the wlevs:application-timestamped Application Assembly Element	C-5
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-25
C-15	Attributes of the wlevs:listener Application Assembly Element.....	C-26
C-16	Attributes of the wlevs:metadata Application Assembly Element	C-27
C-17	Attributes of the wlevs:processor Application Assembly Element.....	C-28
C-18	Attributes of the wlevs:property Application Assembly Element	C-30
C-19	Attributes of the wlevs:source Application Assembly Element	C-30
C-20	Attributes of the wlevs:table Application Assembly Element	C-31
C-21	Attributes of the wlevs:table-source Application Assembly Element	C-32
D-1	Attributes of the binding Component Configuration Element.....	D-17
D-2	Attributes of the database Component Configuration Element	D-30
D-3	Attributes of the group-binding Component Configuration Element.....	D-41
D-4	Attributes of the listeners Component Configuration Element.....	D-46
D-5	Attributes of the param Component Configuration Element	D-52
D-6	Attributes of the params Component Configuration Element.....	D-54
D-7	Attributes of the query Component Configuration Element	D-61
D-8	Attributes of the rule Component Configuration Element.....	D-63
D-9	Attributes of the view Component Configuration Element	D-83
E-1	Attributes of the wlevs:deployment Deployment Element	E-2
F-1	Child Elements of: auth-constraint.....	F-4
F-2	Child Elements of: bdb-config.....	F-5
F-3	Child Elements of: channel-resource-collection	F-8
F-4	Child Elements of: cluster.....	F-9
F-5	Child Elements of: connection-pool-params.....	F-11
F-6	Child Elements of: data-source-params.....	F-15
F-7	Child Elements of: driver-params	F-17
F-8	Child Elements of: debug.....	F-18
F-9	Child Elements of: event-store	F-19
F-10	Child Elements of: exported-jndi-context	F-20

F-11	Child Elements of: jetty	F-21
F-12	Child Elements of: jetty-web-app	F-22
F-13	Child Elements of: jmx	F-23
F-14	Child Elements of: jndi-context.....	F-24
F-15	Child Elements of: log-file	F-25
F-16	Child Elements of: log-stdout	F-26
F-17	Child Elements of: logging-service.....	F-27
F-18	Child Elements of: netio	F-29
F-19	Child Elements of: netio-client.....	F-30
F-20	Child Elements of: rdbms-event-store-provider	F-32
F-21	Child Elements of: rmi	F-33
F-22	Child Elements of: scheduler	F-34
F-23	Child Elements of: server-config	F-35
F-24	Child Elements of: services.....	F-37
F-25	Child Elements of: show-detail-error-message	F-38
F-26	Child Elements of: ssl	F-39
F-27	Child Elements of: transaction-manager	F-41
F-28	Child Elements of: use-secure-connections.....	F-44
F-29	Child Elements of: weblogic-instances	F-44
F-30	Child Elements of: weblogic-rmi-client	F-46
F-31	Child Elements of: work-manager	F-47
F-32	Child Elements of: xa-params	F-47
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

Preface

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

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

Audience

This document is intended for programmers creating Oracle CEP applications.

Documentation Accessibility

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

Accessibility of Code Examples in Documentation

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

Accessibility of Links to External Web Sites in Documentation

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

Access to Oracle Support

Oracle customers have access to electronic support through My Oracle Support. For information, visit <http://www.oracle.com/support/contact.html> or visit <http://www.oracle.com/accessibility/support.html> if you are hearing impaired.

Related Documents

For more information, see the following:

- *Oracle Complex Event Processing Getting Started*
- *Oracle Complex Event Processing Administrator's Guide*
- *Oracle Complex Event Processing Visualizer User's Guide*
- *Oracle Fusion Middleware Java API Reference for Oracle Complex Event Processing*
- *Oracle Complex Event Processing CQL Language Reference*
- *Oracle Complex Event Processing EPL Language Reference*
- *Oracle Database SQL Language Reference* at 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 CEP Forum:
<http://forums.oracle.com/forums/forum.jspa?forumID=820>
- Oracle CEP Samples:
<http://www.oracle.com/technologies/soa/complex-event-processing.html>
- Oracle Event Driven Architecture Suite sample code:
http://www.oracle.com/technology/sample_code/products/event-driven-architecture

Conventions

The following text conventions are used in this document:

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

Part I

Introduction

Part I contains the following chapters:

- [Section 1, "Overview of Creating Oracle CEP Applications"](#)

Overview of Creating Oracle CEP Applications

This section contains information on the following subjects:

- [Section 1.1, "Overview of the Oracle CEP Programming Model"](#)
- [Section 1.2, "Oracle CEP IDE for Eclipse"](#)
- [Section 1.3, "Creating an Oracle CEP Application"](#)
- [Section 1.4, "Configuring Oracle CEP Resource Access"](#)
- [Section 1.5, "Next Steps"](#)

1.1 Overview of the Oracle CEP Programming Model

Because Oracle CEP 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 CEP services, such as configuration management, to those provided by another Oracle product such as Oracle Kodo, to those provided by a third party.

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

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

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

- [Section 1.1.1, "Components of the Oracle CEP Event Processing Network"](#)
- [Section 1.1.2, "Oracle CEP Event Types"](#)
- [Section 1.1.3, "Transmitting Events in the EPN: Stream and Relation Sources and Sinks"](#)

- [Section 1.1.4, "EPN Assembly File"](#)
- [Section 1.1.5, "Component Configuration Files"](#)
- [Section 1.1.5.1, "Accessing Component and Server Configuration Using the ConfigurationPropertyPlaceholderConfigurer Class"](#)
- [Section 1.1.8, "High Availability and Scalability"](#)
- [Section 1.1.7, "Extending the EPN"](#)
- [Section 1.1.6, "How Components Fit Together"](#)
- [Section 1.1.9, "Oracle CEP Application Lifecycle"](#)
- [Section 1.1.10, "Oracle CEP APIs"](#)

1.1.1 Components of the Oracle CEP Event Processing Network

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

Oracle CEP applications and their EPNs are made up of the following basic components:

- [Section 1.1.1.1, "Adapter"](#)
- [Section 1.1.1.2, "Channel"](#)
- [Section 1.1.1.3, "Processor"](#)
- [Section 1.1.1.4, "Event Bean"](#)
- [Section 1.1.1.5, "Spring Bean"](#)
- [Section 1.1.1.6, "Cache"](#)
- [Section 1.1.1.7, "Table"](#)

Each component on the EPN is known as a **stage**. Within the Oracle CEP IDE for Eclipse, each component on the EPN is also known as a **node**.

For more information, see:

- [Section 1.1.1.8, "Nested Stages"](#)
- [Section 1.1.1.9, "Foreign Stages"](#)
- [Section 6.4.1, "Creating Nodes"](#)

1.1.1.1 Adapter

An adapter is a component that provides an interface to incoming data feeds and converts the data into event types that the Oracle CEP application understands.

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

For more information, see:

- [Chapter 7, "Configuring JMS Adapters"](#)
- [Chapter 8, "Configuring HTTP Publish-Subscribe Server Adapters"](#)
- [Chapter 14, "Configuring Custom Adapters"](#)
- [Section 21.2, "Configuring High Availability Adapters"](#)

1.1.1.2 Channel

A channel is a component that represents the physical conduit through which events flow.

Channels connect components that send events with components that receive events. For more information, see [Chapter 9, "Configuring Channels"](#).

1.1.1.3 Processor

A processor is a component that executes user-defined event processing rules against the events offered by channels.

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

- Oracle Continuous Query Language (Oracle CQL)

For more information, see:

- [Chapter 10, "Configuring Oracle CQL Processors"](#)
- *Oracle Complex Event Processing CQL Language Reference*

- Event Processing Language (EPL)

For more information, see:

- [Chapter 11, "Configuring EPL Processors"](#)
- *Oracle Complex Event Processing EPL Language Reference*

Note: Oracle CQL replaces Event Processing Language (EPL) in Oracle CEP 11g Release 1 (11.1.1). Oracle CEP supports EPL for backwards compatibility.

1.1.1.4 Event Bean

An event bean is a user-coded Plain Old Java Object (POJO) bean managed by Oracle CEP.

Event beans are analogous to Spring beans, which are managed by the Spring framework, however they support event-based features, such as the specification of a suspend and resume methods. For more information, see [Chapter 15, "Configuring Custom Event Beans"](#).

1.1.1.5 Spring Bean

A spring bean is a user-coded Plain Old Java Object (POJO) managed by the Spring framework.

These components are not managed by Oracle CEP; for example, you cannot monitor these components, record and playback of events, and so on. If you require this additional functionality for your POJO, consider creating an Oracle CEP event bean instead. For more information, see [Section 16, "Configuring Custom Spring Beans"](#).

1.1.1.6 Cache

A cache is a temporary storage area for events, created exclusively to improve the overall performance of your application and to handle large amounts of historical data.

For more information, see:

- [Section 10.4, "Configuring an Oracle CQL Processor Cache Source"](#)

- [Section 11.3, "Configuring an EPL Processor Cache Source"](#)
- [Chapter 12, "Configuring Caching"](#).

1.1.1.7 Table

A table is a component that connects a relational database table to the EPN as an event data 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 Complex Event Processing CQL Language Reference*.

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 10.3, "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 Complex Event Processing CQL Language Reference*.

In all cases, you must define datasources in the Oracle CEP server `config.xml` file. For more information, see "Configuring Access to a Relational Database" in the *Oracle Complex Event Processing Administrator's Guide*.

Oracle CEP relational database table event sources are pull data sources: that is, Oracle CEP will periodically poll the event source.

1.1.1.8 Nested Stages

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: id="helloworldInput" event-type="HelloWorldEvent" >
      <wlevs:listener>
        <wlevs:processor id="helloworldProcessor">
          <wlevs:listener>
            <wlevs: id="helloworldOutput" event-type="HelloWorldEvent">
              <wlevs:listener>
                <bean class="com.bea.wlevs.example.helloworld.HelloWorldBean"/>
              </wlevs:listener>
            </wlevs:>
          </wlevs:listener>
        </wlevs:processor>
      </wlevs:listener>
    </wlevs:>
  </wlevs:listener>
</wlevs:adapter>

```

For more information, see [Section 6.2.9, "Nested Stages"](#).

1.1.1.9 Foreign Stages

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

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

```
FOREIGN-APPLICATION-NAME: FOREIGN-STAGE-ID
```

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

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

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

```

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

```

```
</wlevs:stream>
```

Example 1–4 Foreign Stage in Application 2

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

The following stages cannot be foreign stages:

- Cache

When creating Oracle CEP applications with foreign stages, you must consider foreign stage dependencies when assembling, deploying, and redeploying your application. For more information, see [Section 24.2.3, "Assembling Applications With Foreign Stages"](#).

1.1.2 Oracle CEP Event Types

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

Events are JavaBean or Java class instances in which each property represents a data item from the event source. Oracle CEP supports a variety of event type implementations.

For more information, see [Chapter 2, "Overview of Oracle CEP Events"](#).

1.1.3 Transmitting Events in the EPN: Stream and Relation Sources and Sinks

All components in an EPN either produce events or consume events. Some components can do both, such as an event bean in the middle of an EPN that receives events from an upstream component, such as an adapter, and then sends events to a downstream component, such as a processor.

Components can act on the events they receive from an upstream component, pass events on to a downstream component, and create new events and insert them into an upstream or downstream channel.

In general, predefined components (such as JMS and HTTP publish-subscribe adapters, channels, and processors) all implement the required interfaces to operate as stream and relation sources and sinks. You can use them as is or configure their behavior using the EPN assembly file.

For custom adapters, event beans, and Spring beans, you must implement the appropriate interfaces to use these components as stream and relation sources and sinks.

This section describes:

- [Section 1.1.3.1, "Streams and Relations"](#)
- [Section 1.1.3.2, "Stream and Relation Sources"](#)
- [Section 1.1.3.3, "Stream and Relation Sinks"](#)
- [Section 1.1.3.4, "Transmitting Events in the EPN: Examples"](#)

For more information, see [Section 1.1.9.6, "User Action: Calling Methods of Stream and Relation Sources and Sinks"](#).

1.1.3.1 Streams and Relations

Oracle CEP models event flow as either streams or relations:

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

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

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

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

For more information, see:

- [Section 9.1.2, "Channels Representing Streams and Relations"](#)
- "Streams and Relations" in the *Oracle Complex Event Processing CQL Language Reference*

1.1.3.2 Stream and Relation Sources

All components that produce events must implement one of the following interfaces:

- [com.bea.wlevs.ede.api.StreamSource](#)
- [com.bea.wlevs.ede.api.RelationSource](#)

com.bea.wlevs.ede.api.StreamSource

The `StreamSource` interface has one method, `setEventSender`; at runtime the event source component is injected with a `com.bea.wlevs.ede.api.StreamSender` instance.

The `StreamSender` instance, in turn, has a `sendInsertEvent` method that the component invokes to actually send events to the next component in the EPN.

The `StreamSender` instance also provides a `sendHeartbeat` method. The `sendHeartBeat` method applies only to application-timestamped channels.

If you enable batching on a channel, then components that send events to that channel must implement the `com.bea.wlevs.ede.api.BatchStreamSender` interface.

The `BatchStreamSender` interface has a `sendInsertEvents` method that the component invokes to send a batch of events of type `insert` downstream to a `BatchStreamSink`. For more information, see [Section 9.1.6, "Batch Processing Channels"](#).

com.bea.wlevs.ede.api.RelationSource

The `RelationSource` interface has one method, `setEventSender`; at runtime the event source component is injected with a `com.bea.wlevs.ede.api.RelationSender` instance.

The `RelationSender` instance, in turn, has `sendUpdateEvent` and `sendDeleteEvent` methods that the component invokes to actually send events to the next component in the EPN.

When you configure a channel as a relation, the channel attribute `primary-key` determines event identity. For more information, see [Section 9.1.2.2, "Channels as Relations"](#).

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 discarded if an event with the given primary key is not in the relation.

If you enable batching on a channel you configure as a relation, then components that send events to that channel must implement the `com.bea.wlevs.ede.api.BatchRelationSender` interface. The `BatchRelationSender` interface has a `sendEvents` method that the component invokes to send a batch of events of type `insert`, `update`, and `delete` downstream to a `BatchRelationSink`. For more information, see [Section 9.1.6, "Batch Processing Channels"](#).

1.1.3.3 Stream and Relation Sinks

All components that consume events must implement one of the following interfaces:

- [com.bea.wlevs.ede.api.StreamSink](#)
- [com.bea.wlevs.ede.api.RelationSink](#)

com.bea.wlevs.ede.api.StreamSink

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

If you enable batching on a channel, then components that receive events from that channel must implement the `com.bea.wlevs.ede.api.BatchStreamSink` interface. The `BatchStreamSink` interface has an `onInsertEvents` call-back method for receiving a batch of events of type `insert`. Events of type `insert` are used to model streams, as streams are append-only. In the case of single event, call-back method of `StreamSink` will be invoked. For more information, see [Section 9.1.6, "Batch Processing Channels"](#).

com.bea.wlevs.ede.api.RelationSink

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

If you enable batching on a channel you configure as a relation, then components that receive events from that channel must implement the `com.bea.wlevs.ede.api.BatchRelationSink` interface. The `BatchRelationSink` interface has an `onEvents` call-back method for receiving a

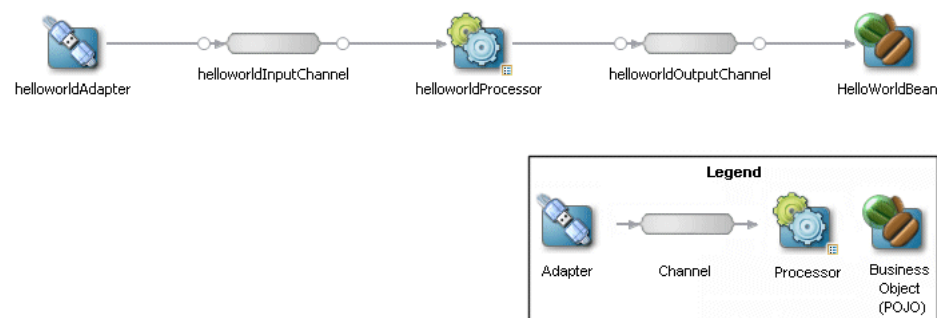
batch of events for each event type of insert, update, and delete. If there is no event of a particular kind, then the corresponding `Collection` will be an empty `Collection`. For more information, see [Section 9.1.6, "Batch Processing Channels"](#).

1.1.3.4 Transmitting Events in the EPN: Examples

The following example is based on the HelloWorld example that [Figure 1–1](#) shows. It is modified here to illustrate:

- [Creating the HelloWorldEvent as a Tuple Event](#)
- [Sending Events from the HelloWorldAdapter as a StreamSource](#)
- [Receiving Events in the HelloWorldBean as a StreamSink](#)
- [Sending Events from the HelloWorldBean as a StreamSource](#)

Figure 1–1 The HelloWorld Example Event Processing Network



For more information, see:

- [Chapter 14, "Configuring Custom Adapters"](#)
- [Chapter 15, "Configuring Custom Event Beans"](#)
- [Chapter 16, "Configuring Custom Spring Beans"](#)
- "HelloWorld Example" in the *Oracle Complex Event Processing Getting Started*

Creating the HelloWorldEvent as a Tuple Event

First, create the tuple event in the EPN assembly file:

```
<wlevs:event-type-repository>
  <wlevs:event-type type-name="HelloWorldEvent">
    <wlevs:properties>
      <wlevs:property name="message" type="char" length="1000"/>
    </wlevs:properties>
  </wlevs:event-type>
</wlevs:event-type-repository>
```

Sending Events from the HelloWorldAdapter as a StreamSource

Modify the `HelloWorldAdapter` to inject an `EventTypeRepository` instance at runtime by adding a `setEventTypeRepository` method using the `@Service` annotation:

```
@Service
public void setEventTypeRepository(EventTypeRepository etr) {
    etr_ = etr;
}
```

Modify the HelloWorldAdapter's generateHelloMessage method to use the EventTypeRepository instance to instantiate a HelloWorldEvent, then configure the event's properties, and send the event to the helloWorldInputChannel using the StreamSender instance that Oracle CEP server injected at runtime:

```
private void generateHelloMessage() {
    String message = this.message + dateFormat.format(new Date());

    EventType type = etr_.getEventType("HelloWorldEvent");
    EventProperty messageProp = type.getProperty("message");
    Object event = type.createEvent();

    messageProp.setValue(event, message);

    eventSender.sendInsertEvent(event);
}
```

Receiving Events in the HelloWorldBean as a StreamSink

Modify the HelloWorldBean to inject an EventTypeRepository instance at runtime using the @Service annotation:

```
@Service
public void setEventTypeRepository(EventTypeRepository etr) {
    etr_ = etr;
}
```

Modify the HelloWorldBean's onInsertEvent method use the EventTypeRepository instance to get the type of the received insert event and get the event's property.

```
public void onInsertEvent(Object event) {
    EventType eventType = etr_.getEventType(event);
    String prop = (String)eventType.getPropertyValue(event, "message");

    System.out.println("Tuple Message: " + prop);
}
```

Sending Events from the HelloWorldBean as a StreamSource

Imagine that the HelloWorldBean is connected to another component and you want to send events from the HelloWorldBean to this downstream component.

First, modify the HelloWorldBean class to implement the StreamSource interface and add the required setEventSender method:

```
public class HelloWorldBean implements StreamSource, StreamSink {

    private StreamSender streamSender_;

    public void setEventSender(StreamSender sender) {
        streamSender_ = sender;
    }
    ...
}
```

Modify the HelloWorldBean to inject an EventTypeRepository instance at runtime by adding a setEventTypeRepository method using the @Service annotation:

```
@Service
public void setEventTypeRepository(EventTypeRepository etr) {
```

```

    etr_ = etr;
}

```

Modify the `HelloWorldBean` to use the `EventTypeRepository` instance to instantiate a `HelloWorldEvent`, then configure the event's properties, and send the event to the downstream component using the `StreamSender` instance that Oracle CEP server injected at runtime:

```

private void generateHelloMessage() {
    String message = this.message + dateFormat.format(new Date());

    EventType type = etr_.getEventType("HelloWorldEvent");
    EventProperty messageProp = type.getProperty("message");
    Object event = type.createEvent();

    messageProp.setValue(event, message);

    eventSender.sendInsertEvent(event);
}

```

1.1.4 EPN Assembly File

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

The `spring-wlevs-v11_1_1_3.xsd` schema file describes the structure of EPN assembly files.

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

```

<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:osgi="http://www.springframework.org/schema/osgi"
    xmlns:wlevs="http://www.bea.com/ns/wlevs/spring"
    xsi:schemaLocation="
        http://www.springframework.org/schema/beans
        http://www.springframework.org/schema/beans/spring-beans.xsd
        http://www.springframework.org/schema/osgi
        http://www.springframework.org/schema/osgi/spring-osgi.xsd
        http://www.bea.com/ns/wlevs/spring
        http://www.bea.com/ns/wlevs/spring/spring-wlevs-v11_1_1_3.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 CEP features, you specify some configuration in the EPN assembly file and some in the component configuration file.

For more information, see:

- [Section 4.3, "Creating EPN Assembly Files"](#)
- [Section 1.1.5, "Component Configuration Files"](#)
- [Section 1.1.6, "How Components Fit Together"](#)
- [Appendix B.1, "EPN Assembly Schema spring-wlevs-v11_1_1_3.xsd"](#)

1.1.5 Component Configuration Files

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

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

- `wlevs_base_config.xsd`: Defines common elements that are shared between application configuration files and the server configuration file
- `wlevs_eventstore_config.xsd`: Defines event store-specific elements.
- `wlevs_diagnostic_config.xsd`: Defines diagnostic elements.

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

```

<?xml version="1.0" encoding="UTF-8"?>
<n1:config xmlns:n1="http://www.bea.com/ns/wlevs/config/application"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
    <processor>
        <name>helloworldProcessor</name>
        <rules>
            <query id="helloworldRule">
                <![CDATA[ select * from helloworldInputChannel [Now] ]]>
            </query>
        </rules>
    </processor>
    <channel>
        <name>helloworldInputChannel</name>
        <max-size>10000</max-size>
        <max-threads>2</max-threads>
    </channel>
</config>

```

```

</channel>
<channel>
  <name>helloworldOutputChannel</name>
  <max-size>10000</max-size>
  <max-threads>2</max-threads>
</channel>
</n1:config>

```

For more information, see:

- [Section 1.1.5.1, "Accessing Component and Server Configuration Using the ConfigurationPropertyPlaceholderConfigurer Class"](#)
- [Section 1.1.6, "How Components Fit Together"](#)
- [Section 4.4, "Creating Component Configuration Files"](#)
- [Appendix B.2, "Component Configuration Schema wlevs_application_config.xsd"](#)

1.1.5.1 Accessing Component and Server Configuration Using the ConfigurationPropertyPlaceholderConfigurer Class

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

For example, given the element that [Example 1–5](#) shows, you can refer to the `netio` element `port` value as [Example 1–6](#) shows.

Example 1–5 netio Element

```

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

```

Example 1–6 Accessing the netio Element port Value

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

To use this feature, insert a `ConfigurationPropertyPlaceholderConfigurer` bean in the application context configuration file of your application bundle as [Example 1–7](#) shows.

Example 1–7 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 Complex Event Processing*.

For more information on accessing property files, see [Section 4.7.3, "How to Add a Property File to an Oracle CEP Project"](#).

1.1.6 How Components Fit Together

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

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

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

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

- The compiled application Java code of the business logic POJO.
- Component configuration files. Each processor is required to have a configuration file, although adapters and streams do not need to have a configuration file if the default configuration is adequate and you do not plan to monitor these components.
- The EPN assembly file.
- A `MANIFEST.MF` file with some additional OSGi entries.

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

For more information, see [Chapter 24, "Assembling and Deploying Oracle CEP Applications"](#).

1.1.7 Extending the EPN

In addition to the components that Oracle CEP provides, you can extend the EPN by creating custom adapters, event beans, and Spring beans.

You can also extend the EPN using the Oracle Java data cartridge and Oracle Spatial to incorporate Java and Oracle Spatial operations in your Oracle CQL queries.

You can also expose Oracle CEP applications as Web services and consume Web services in your Oracle CEP application.

For more information, see [Part IV, "Extending the Oracle CEP Event Processing Network"](#)

1.1.8 High Availability and Scalability

When designing an Oracle CEP application, it is a best practice to consider high availability and scalability early in the design process.

Oracle CEP provides a range of high availability options that allow you to declaratively configure the quality of service your application requires for recovering from server failures.

Oracle CEP also provides a variety of features that allow you to scale your application to accommodate increasing event rates.

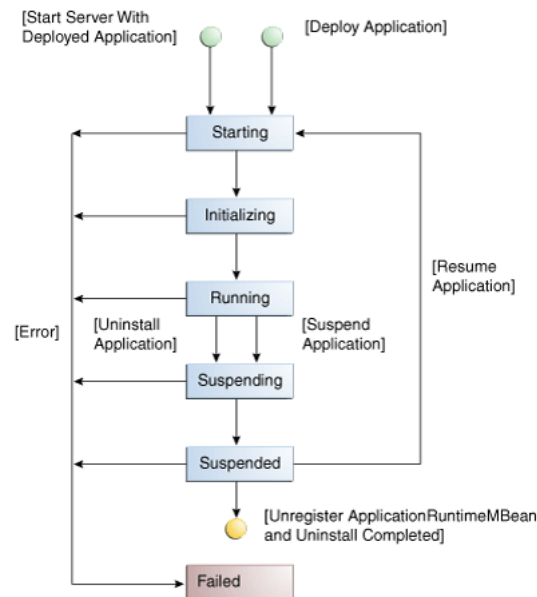
For more information, see:

- [Part V, "Developing Applications for High Availability"](#)
- [Part VI, "Developing Applications for Scalability"](#)

1.1.9 Oracle CEP Application Lifecycle

Figure 1–2 shows a state diagram for the Oracle CEP application lifecycle. In this diagram, the state names (STARTING, INITIALIZING, RUNNING, SUSPENDING, SUSPENDED, and FAILED) correspond to the `ApplicationRuntimeMBean` method `getState` return values. These states are specific to Oracle CEP; they are not OSGi bundle states.

Figure 1–2 Oracle CEP Application Lifecycle State Diagram



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

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

- [Section 1.1.9.1, "User Action: Installing an Application or Start the Server With Application Already Deployed"](#)
- [Section 1.1.9.2, "User Action: Suspend Application"](#)
- [Section 1.1.9.3, "User Action: Resume Application"](#)
- [Section 1.1.9.4, "User Action: Uninstall Application"](#)
- [Section 1.1.9.5, "User Action: Update Application"](#)
- [Section 1.1.9.6, "User Action: Calling Methods of Stream and Relation Sources and Sinks"](#)

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

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

- [Section 1.1.10, "Oracle CEP APIs"](#)
- [Appendix I, "Oracle CEP Metadata Annotation Reference"](#)
- *Oracle Fusion Middleware Java API Reference for Oracle Complex Event Processing*

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

Oracle CEP performs the following actions:

1. Oracle CEP installs the application as an OSGI bundle. OSGI resolves the imports and exports, and publishes the service.
2. Oracle CEP creates beans (for both standard Spring beans and those that correspond to the Oracle CEP tags in the EPN assembly file). For each bean, Oracle CEP:
 - Sets the properties on the Spring beans. The `<wlevs:instance-property>` values are set on adapters and event-beans.
 - Injects appropriate dependencies into services specified by `@Service` or `@ServiceReference` annotations.
 - Injects appropriate dependencies into static configuration properties.
 - Calls the `InitializingBean.afterPropertiesSet` method.
 - Calls configuration callbacks (`@Prepare`, `@Activate`) on Spring beans as well as factory-created stages.
For more information, see [Section 1.4, "Configuring Oracle CEP Resource Access"](#).
3. Application state is now `INITIALIZING`.
4. Oracle CEP registers the MBeans.
5. Oracle CEP calls the `ActivatableBean.afterConfigurationActive` method on all `ActivatableBeans`.
6. Oracle CEP calls the `ResumableBean.beforeResume` method on all `ResumableBeans`.
7. For each bean that implements `RunnableBean`, Oracle CEP starts it running in a thread.
8. Application state is now `RUNNING`.

1.1.9.2 User Action: Suspend Application

Oracle CEP performs the following actions:

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

1.1.9.3 User Action: Resume Application

Oracle CEP performs the following actions:

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

1.1.9.4 User Action: Uninstall Application

Oracle CEP performs the following actions:

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

1.1.9.5 User Action: Update Application

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

See:

- [Section 1.1.9.4, "User Action: Uninstall Application"](#)
- [Section 1.1.9.1, "User Action: Installing an Application or Start the Server With Application Already Deployed"](#)

1.1.9.6 User Action: 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:

- [Section 1.1.9.1, "User Action: Installing an Application or Start the Server With Application Already Deployed"](#)
- [Section 1.1.3, "Transmitting Events in the EPN: Stream and Relation Sources and Sinks"](#)

1.1.10 Oracle CEP APIs

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

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

- `Adapter`—Adapters must implement this interface.

- `AdapterFactory`—Adapter factories must implement this interface.
- Component life cycle interfaces—If you want some control over the life cycle of the component you are programming, then your component should implement one or more of the following interfaces:
 - `ActivatableBean`—Use if you want to run some code after all dynamic configuration has been set and the event processing network has been activated. Implement the `afterConfigurationActive` method.
 - `DisposableBean`—Use if you want to release resources when the application is undeployed. Implement the `destroy` method in your component code.
 - `InitializingBean`—Use if you require custom initialization after Oracle CEP has set all the properties of the component. Implement the `afterPropertiesSet` method.
 - `ResumableBean`—Use if you want to perform some task, such as acquire or configure resources, before the component resumes work.
 - `RunnableBean`—Use if you want the component to be run in a thread.

The Spring framework implements similar bean life cycle interfaces; however, the equivalent Spring interfaces do not allow you to manipulate beans that were created by factories, while the Oracle CEP interfaces do.
 - `SuspendableBean`—Use if you want to suspend resources or stop processing events when the event processing network is suspended. Implement the `suspend` method.

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

- `EventBuilder`—Use to create events whose Java representation does not expose the necessary setter and getter methods for its properties. If your event type is represented with a `JavaBean` with all required getter and setter methods, then you do not need to create an `EventBuilder`.
- `EventBuilderFactory`—Factory for creating `EventBuilders`.
- `StreamSink`—Components that want to receive events as an Oracle CEP stream must implement this interface. An Oracle CEP stream has the following characteristics:
 - Append-only, that is, events are always appended to the end of the stream.
 - Unbounded and generally need a window to be defined before it can be processed.
 - Events have non-decreasing time-stamps.

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

- `StreamSource`—Components that send events modeling an Oracle CEP stream, such as adapters, must implement this interface. The interface has a `setEventSender` method for setting the `StreamSender`, which actually sends the event to the next component in the network.
- `RelationSink`—Components that want to receive events modeling an Oracle CEP relation must implement this interface. An Oracle CEP relation has the following characteristics:
 - Supports events that insert, delete, and update its content.

- Is always known at an instant time.
- Events have non-decreasing time-stamps.

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

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

For more information, see:

- For the full reference documentation for all classes and interfaces, see *Oracle Fusion Middleware Java API Reference for Oracle Complex Event Processing*.
- For sample Java code that uses these APIs, see:
 - [Chapter 14, "Configuring Custom Adapters"](#)
 - [Chapter 15, "Configuring Custom Event Beans"](#)
 - [Chapter 16, "Configuring Custom Spring Beans"](#)
 - "Oracle CEP Samples" in the *Oracle Complex Event Processing Getting Started*.
- [Section 1.4, "Configuring Oracle CEP Resource Access"](#) for information on using Oracle CEP annotations and deployment XML to configure resource injection.

1.2 Oracle CEP IDE for Eclipse

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

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

- Project creation wizards and templates to quickly get started building event driven applications.
- Advanced editors for source files including Java and XML files common to Oracle CEP applications.
- Integrated server management to seamlessly start, stop, and deploy to Oracle CEP server instances all from within the IDE.
- Integrated debugging.
- Event Processing Network (EPN) visual design views for orienting and navigating in event processing applications.
- Integrated support for the Oracle CEP Visualizer so you can use the Oracle CEP Visualizer from within the IDE.

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

For detailed instructions on installing and using the IDE, see [Chapter 3, "Overview of the Oracle CEP IDE for Eclipse"](#).

1.3 Creating an Oracle CEP Application

The following procedure shows the *suggested* start-to-finish steps to create an Oracle CEP application. Although it is not required to program and configure the various

components in the order shown, the procedure shows a typical and logical flow recommended by Oracle.

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

To create an Oracle CEP application:

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

2. Create an Oracle CEP project using the Oracle CEP IDE for Eclipse.

For more information, see [Chapter 4, "Oracle CEP IDE for Eclipse Projects"](#).

3. Design your event processing network (EPN).

Using the Oracle CEP 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 6, "Oracle CEP IDE for Eclipse and the Event Processing Network"](#)
- [Section 4.3, "Creating EPN Assembly Files."](#)
- [Section 4.4, "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 2.1.4, "Creating Oracle CEP Event Types"](#)

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

- [Section 1.1.10, "Oracle CEP APIs"](#)
- [Section 1.4, "Configuring Oracle CEP Resource Access"](#)
- [Chapter 7, "Configuring JMS Adapters"](#)
- [Chapter 8, "Configuring HTTP Publish-Subscribe Server Adapters"](#)
- [Chapter 14, "Configuring Custom Adapters"](#)
- [Chapter 15, "Configuring Custom Event Beans"](#)

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

See:

- [Chapter 10, "Configuring Oracle CQL Processors"](#)

- [Chapter 11, "Configuring EPL Processors"](#)
- 7. 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 Complex Event Processing CQL Language Reference*
- *Oracle Complex Event Processing EPL Language Reference*

Note: Oracle CQL replaces Event Processing Language (EPL) in Oracle CEP 11g Release 1 (11.1.1). Oracle CEP 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 9, "Configuring 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 12, "Configuring Caching."](#)
- 10. Optionally, use the Oracle CEP server log subsystem to write log messages from your application to the Oracle CEP server log:

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

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

For more information, see:

- "Configuring Logging and Debugging for Oracle CEP" in the *Oracle Complex Event Processing Administrator's Guide*
- [Section 24.4, "Managing Log Message Catalogs"](#)
- "How to Configure Component Logging" in the *Oracle Complex Event Processing Visualizer User's Guide*
- "How to View Console Output" in the *Oracle Complex Event Processing Visualizer User's Guide*

See [Section 1.5, "Next Steps"](#) for the list of steps you should follow after you have completed programming your application, such as packaging and deploying.

1.4 Configuring Oracle CEP Resource Access

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

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

Oracle CEP supports the following types of resource access:

- [Section 1.4.1, "Static Resource Injection"](#)
- [Section 1.4.2, "Dynamic Resource Injection"](#)
- [Section 1.4.3, "Dynamic Resource Lookup Using JNDI"](#)

See [Section 1.4.4, "Understanding Resource Name Resolution"](#) for information on resource name resolution.

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

In the following sections, consider the example resource that [Example 1–8](#) shows. This is a data source resource named `StockDS` that you specify in the Oracle CEP server `config.xml` file.

Example 1–8 Sample Resource: Data Source StockDS

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

1.4.1 Static Resource Injection

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

You can configure static resource injection using:

- [Section 1.4.1.1, "Static Resource Names"](#)
- [Section 1.4.1.2, "Dynamic Resource Names"](#)

1.4.1.1 Static Resource Names

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

To configure static resource injection using static resource names at design time, you use the standard `javax.annotation.Resource` annotation as [Example 1–9](#) shows.

To override design time configuration at deploy time, you use Oracle CEP assembly file XML as [Example 1–10](#) shows.

In [Example 1–9](#) and [Example 1–10](#), the resource name `StockDs` exactly matches the name of the data source in the Oracle CEP server `config.xml` file as [Example 1–8](#) shows.

Example 1–9 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 1–10 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 1–11](#) shows. Similarly, in this case, the `wlevs:resource` element name attribute is not needed as [Example 1–12](#).

Example 1–11 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 1–12 Static Resource Injection Using Static Resource Names: XML

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

1.4.1.2 Dynamic Resource Names

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

To add a dynamic resource name to a component, such as an adapter or POJO, you must first specify custom configuration for your component that contains the resource name as [Example 1–13](#) shows.

Example 1–13 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 1–14](#) shows.

To override design time configuration at deploy time, you use Oracle CEP assembly file XML as [Example 1–15](#) shows.

Example 1–14 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 1–15 Static Resource Injection Using Dynamic Resource Names: XML

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

1.4.2 Dynamic Resource Injection

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

To configure dynamic resource injection at design time, you use the standard `javax.annotation.Resource` annotation as [Example 1–16](#) shows.

Example 1–16 Dynamic Resource Injection: Annotations

```
import javax.annotations.Resource;

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

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

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

- [Section I.2, "com.bea.wlevs.configuration.Activate"](#)
- [Section I.3, "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.

1.4.3 Dynamic Resource Lookup Using JNDI

Oracle CEP supports the use of JNDI to look up resources dynamically as [Example 1–17](#).

Example 1–17 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 1–17](#), the JNDI name `StockDs` must exactly match the name of the data source in the Oracle CEP server `config.xml` file as [Example 1–8](#) shows.

Note: You must disable security when starting the Oracle CEP server in order to use JNDI. Oracle does not recommend the use of JNDI for this reason.

For more information, see "Configuring Security for Oracle CEP" in the *Oracle Complex Event Processing Administrator's Guide*.

1.4.4 Understanding Resource Name Resolution

Oracle CEP 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
JNDI	The names registered in the server's JNDI registry	Matching

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

1.5 Next Steps

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

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

See [Section 24.2, "Assembling an Oracle CEP Application."](#)

- Start Oracle CEP.

See:

- [Section 5.3.1, "How to Start a Local Oracle CEP Server"](#)
- [Section 5.3.2, "How to Stop a Local Oracle CEP Server"](#)

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

See:

- [Section 5.2, "Creating Oracle CEP Servers"](#)
- [Section 5.3, "Managing Oracle CEP Servers"](#)
- [Section 24.4, "Managing Log Message Catalogs"](#)
- "Understanding Oracle CEP Configuration" in the *Oracle Complex Event Processing Administrator's Guide*

- Deploy the application to Oracle CEP.

See [Section 24.5, "Deploying Oracle CEP Applications."](#)

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

The Oracle CEP load generator is a testing tool that you can use to test your application, in particular its rules. This testing tool can temporarily replace the adapter component in your application, for testing purposes. For details, see [Chapter 25, "Testing Applications With the Load Generator and csvgen Adapter."](#)

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

See also [Section 5.4, "Debugging an Oracle CEP Application Running on an Oracle CEP Server"](#).

Overview of Oracle CEP Events

This section describes Oracle CEP events, including:

- [Section 2.1, "Oracle CEP Event Types"](#)
- [Section 2.2, "Creating an Oracle CEP Event Type as a JavaBean"](#)
- [Section 2.3, "Creating an Oracle CEP Event Type as a Tuple"](#)
- [Section 2.4, "Creating an Oracle CEP Event Type as a Java Class"](#)
- [Section 2.5, "Creating an Oracle CEP Event Type as a java.util.Map"](#)
- [Section 2.6, "Using an Event Type Builder Factory"](#)
- [Section 2.7, "Accessing the Event Type Repository"](#)
- [Section 2.8, "Sharing Event Types Between Application Bundles"](#)

2.1 Oracle CEP Event Types

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

Events are JavaBean or Java class instances in which each property represents a data item from the event source. Oracle CEP supports the following event type implementations:

- **JavaBean:** an event type based on a Java Bean class.
For more information, see [Section 2.2, "Creating an Oracle CEP Event Type as a JavaBean"](#).
- **Tuple:** an event type you create and register declaratively in the EPN assembly file.
For more information, see [Section 2.3, "Creating an Oracle CEP Event Type as a Tuple"](#).
- **Java Class:** an event type based on a Java class.
For more information, see:
 - [Section 2.4, "Creating an Oracle CEP Event Type as a Java Class"](#)
 - [Section 2.6, "Using an Event Type Builder Factory"](#)

- **java.util.Map**: an event type based on an instance of `java.util.Map`.
For more information, see [Section 2.5, "Creating an Oracle CEP Event Type as a java.util.Map"](#).

Oracle recommends that you define your events using the JavaBean (or Java class and factory) approach. Doing so allows you greater flexibility to deal with event types as part of your application logic and can simplify integration with existing systems.

Alternatively, you can specify the properties of the event type declaratively in the EPN assembly file using `wlevs:property` tags, in which case Oracle CEP will use a `java.util.Map` or `tuple`. `tuple` is the default and is similar to the Oracle CQL `tuple`. It essentially provides the user with an optimized implementation, however, the user must always set and get its value using the `EventTypeRepository` APIs. This approach is best used for quick prototyping or when the application developer does not need to deal with JavaBean events as part of the application logic or due to integration to some legacy system.

For more information, see:

- [Section 2.1.1, "Event Type Instantiation and Immutability"](#)
- [Section 2.1.2, "Event Type and Serialization"](#)
- [Section 2.1.3, "Event Type Data Types"](#)
- [Section 2.1.4, "Creating Oracle CEP Event Types"](#)
- [Section 2.7, "Accessing the Event Type Repository"](#)

2.1.1 Event Type Instantiation and Immutability

In Oracle CEP, events are conceptually immutable. Once an event is instantiated (created and initialized), the Oracle CEP server will not change it and application developers should not change it, either.

In Oracle CEP, an event can be instantiated either by the application developer or by the Oracle CEP server. For example:

- An application developer can instantiate an event in an inbound Spring-bean, event-bean, or adapter.
- Oracle CEP server can instantiate an event when a CQL processor outputs an event to a downstream component.

The Oracle CEP server uses the following procedure to instantiate an event for a JavaBean event type:

- It invokes an empty-argument public constructor for the Java class.
- It invokes a public setter method following JavaBean conventions for each event property.

Note that even though the event is conceptually immutable, setter methods must be available in this case. The Oracle CEP server will invoke these setter methods only once during event initialization.

If a JavaBean event is only being created by the application developer, then making the Java class immutable can help improve performance. A truly immutable bean is read only (provides only getters) and has public constructors with arguments that satisfy immutability. Note that immutability is not possible if the event is being output from a CQL processor, but it is possible if the event is used only as input.

The Oracle CEP server uses the following procedure to instantiate an event for a Java class event type:

- It acquires an instance of the `com.bea.wlevs.ede.api.EventBuilderFactory` you associate with the event type by calling the factory's `createBuilder` method.
- It instantiates an event by calling the `createEvent` method.

Note that again, even though the event is conceptually immutable, setter methods must be available in this case. And, again, the Oracle CEP server will invoke these setter methods only once, indirectly, when the factory initializes the event.

2.1.2 Event Type and Serialization

In general, Oracle CEP events need not be serializable.

However, if you are caching events in Oracle Coherence, then events *must* be serializable.

2.1.3 Event Type Data Types

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

- [Section 2.1.3.1, "Event Types Specified as JavaBean or Java Class"](#)
- [Section 2.1.3.2, "Event Types Specified as `java.util.Map`"](#)
- [Section 2.1.3.3, "Event Types Specified as a Tuple"](#)
- [Section 2.1.3.4, "Event Types for use With a Database Table Source"](#)
- [Section 2.1.3.5, "Event Types for use With the `csvgen` Adapter"](#)

For more information, see:

- "Datatypes" in the *Oracle Complex Event Processing CQL Language Reference*
- "Overview of the EPL Language" in the *Oracle Complex Event Processing EPL Language Reference*

2.1.3.1 Event Types Specified as JavaBean or Java Class

When you define an event type as a JavaBean or Java class, you may use any Java type for its properties.

Using the Oracle Java data cartridge, you may also combine JavaBean or Java class event types with other events types. [Example 2-1](#) shows a tuple event type `Student` that defines its `address` property as Java class event type `Address`.

Example 2-1 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>
```

For more information, see

- [Section 2.2, "Creating an Oracle CEP Event Type as a JavaBean"](#)
- [Section 2.4.1, "How to Create an Oracle CEP Event Type as a Java Class Manually"](#)
- [Section 2.8, "Sharing Event Types Between Application Bundles"](#)
- "Oracle Java Data Cartridge" in the *Oracle Complex Event Processing CQL Language Reference*

2.1.3.2 Event Types Specified as `java.util.Map`

When you specify the properties of the event type declaratively in the EPN assembly file as a `java.util.Map`, you may use any Java type for its properties. However, you specify the event type 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 2-2](#) shows how to use these types:

Example 2-2 Specifying Java Data Types for `java.util.Map` Event Type Properties

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

For more information, see [Section 2.5.1, "How to Create an Oracle CEP Event Type as a `java.util.Map`"](#).

2.1.3.3 Event Types Specified as a Tuple

When you specify the properties of the event type declaratively in the EPN assembly file as a tuple, you may use any CQL primitive types or Java types.

[Example 2-3](#) shows the use of different types:

Example 2-3 Specifying `com.bea.welvs.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 2.3, "Creating an Oracle CEP Event Type as a Tuple"](#).

2.1.3.4 Event Types for use With a Database Table Source

When you specify the properties of an event type (as any of a `JavaBean`, `Java class`, `java.util.Map`, or `tuple`) for use with a relational database table, you must observe the following additional JDBC type restrictions:

- [Table 10–2, "EPN Assembly File event-type Element Property Attributes"](#)
- [Table 10–3, "SQL Column Types and Oracle CEP Type Equivalents"](#)

For more information, see:

- [Section 10.3, "Configuring an Oracle CQL Processor Table Source"](#)
- [Section 2.1.3, "Event Type Data Types"](#)

2.1.3.5 Event Types for use With the csvgen Adapter

When you specify the properties of an event type (as any of a `JavaBean`, `Java class`, `java.util.Map`, or `tuple`) for use with the `csvgen` adapter, you may only use the data types that [Table 2–1](#) describes.

Table 2–1 *csvgen Adapter Types*

Type	Usage
<code>char</code>	Single or multiple character values. Use for both <code>char</code> and <code>java.lang.String</code> values. Optionally, you may use the <code>length</code> attribute to specify the maximum length of the <code>char</code> value as Example 2–3 shows for the property with name <code>id</code> . The default length is 256 characters. If you need more than 256 characters you should specify an adequate length.
<code>int</code>	Numeric values in the range that <code>java.lang.Integer</code> specifies.
<code>long</code>	Numeric values in the range that <code>java.lang.Long</code> specifies.
<code>double</code>	Numeric values in the range that <code>java.lang.Double</code> specifies.

For more information, see:

- [Section 25.4, "Creating a Data Feed File"](#)
- [Section 2.1.3.1, "Event Types Specified as JavaBean or Java Class"](#)
- [Section 2.1.3.3, "Event Types Specified as a Tuple"](#)

2.1.4 Creating Oracle CEP Event Types

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

This section describes:

- [Section 2.2, "Creating an Oracle CEP Event Type as a JavaBean"](#)
- [Section 2.3, "Creating an Oracle CEP Event Type as a Tuple"](#)
- [Section 2.4, "Creating an Oracle CEP Event Type as a Java Class"](#)
- [Section 2.5, "Creating an Oracle CEP Event Type as a java.util.Map"](#)

- [Section 2.6, "Using an Event Type Builder Factory"](#)
- [Section 2.7, "Accessing the Event Type Repository"](#)
- [Section 2.8, "Sharing Event Types Between Application Bundles"](#)

For more information, see [Section 2.1, "Oracle CEP Event Types"](#).

2.2 Creating an Oracle CEP Event Type as a JavaBean

You can create and register an Oracle CEP event type as a JavaBean. This is the preferred approach.

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

Oracle recommends that, if possible, you make your event type JavaBeans immutable to improve performance. For more information, see [Section 2.1.1, "Event Type Instantiation and Immutability"](#).

When you design your event, you must restrict your design to the even data types that [Section 2.1.3.1, "Event Types Specified as JavaBean or Java Class"](#) describes.

This topic describes:

- [Section 2.2.1, "How to Create an Oracle CEP Event Type as a JavaBean Using the Event Type Repository Editor"](#)
- [Section 2.2.2, "How to Create an Oracle CEP Event Type as a JavaBean Manually"](#)

2.2.1 How to Create an Oracle CEP Event Type as a JavaBean Using the Event Type Repository Editor

This procedure describes how to create and register an Oracle CEP event type as a JavaBean using the Oracle CEP IDE for Eclipse event type repository editor. For more information about the Oracle CEP IDE for Eclipse, see [Example 3, "Overview of the Oracle CEP IDE for Eclipse"](#).

Alternatively, you can create and register your event type as a JavaBean manually (see [Section 2.2.2, "How to Create an Oracle CEP Event Type as a JavaBean Manually"](#)).

To create an Oracle CEP event type as a Java bean using the event type repository editor:

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

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

Oracle recommends that, if possible, you make your event type JavaBeans immutable to improve performance. For more information, see [Section 2.1.1, "Event Type Instantiation and Immutability"](#).

When you design your event, you must restrict your design to the even data types that [Section 2.1.3.1, "Event Types Specified as JavaBean or Java Class"](#) describes.

[Example 2-5](#) shows the `MarketEvent` which is implemented by the `com.bea.wlevs.example.algotrading.event.MarketEvent` class.

Example 2-4 MarketEvent Class

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

import java.util.Date;

public final class MarketEvent {
    private final Long timestamp;

    private final String symbol;

    private final Double price;

    private final Long volume;

    private final Long latencyTimestamp;

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

    public Double getPrice() {
        return price;
    }

    public String getSymbol() {
        return symbol;
    }

    public Long getTimestamp() {
        return timestamp;
    }

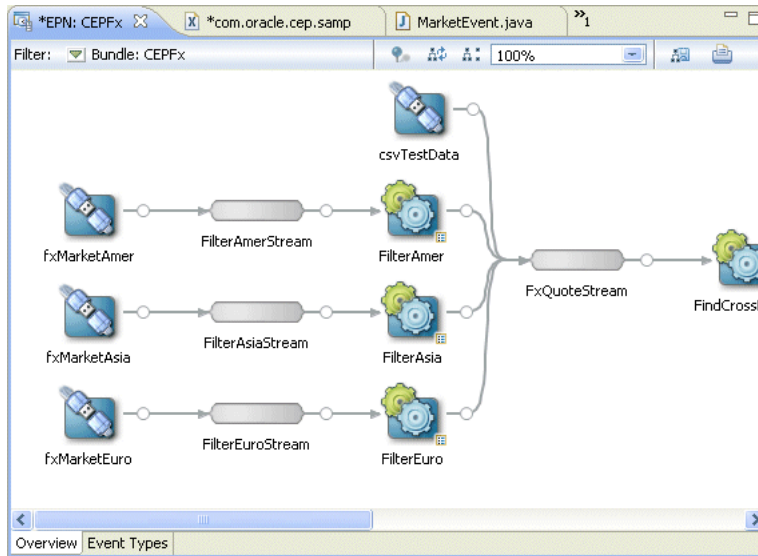
    public Long getLatencyTimestamp() {
        return latencyTimestamp;
    }

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

2. Compile the JavaBean that represents your event type.
3. Open the EPN in the Oracle CEP IDE for Eclipse.

The EPN editor opens as [Figure 2-1](#) shows.

Figure 2–1 EPN Editor

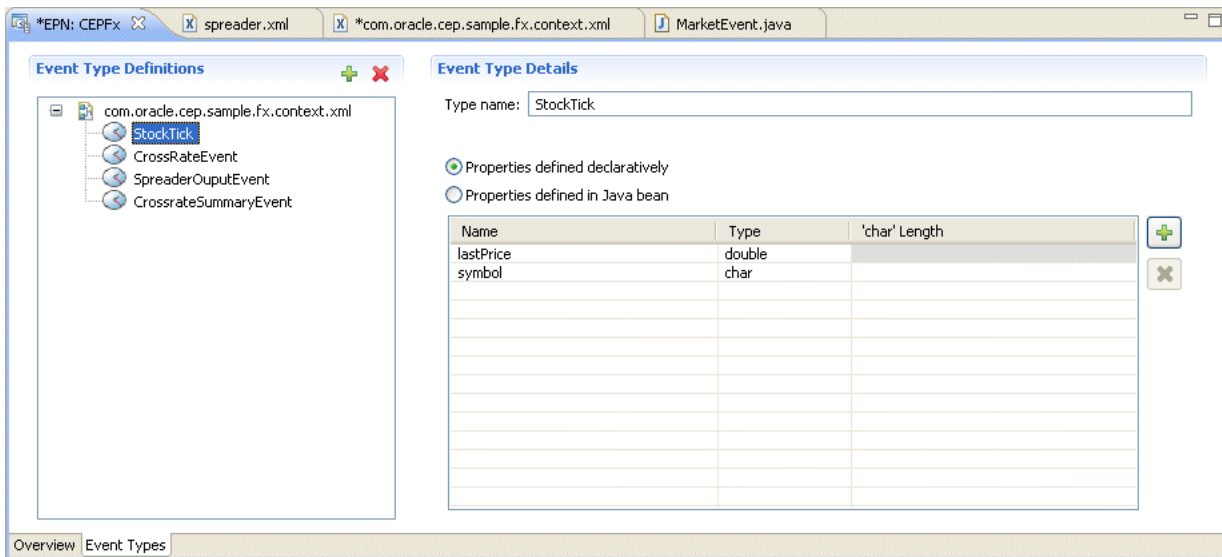


For more information, see [Section 6.1, "Opening the EPN Editor"](#).

4. Click the **Event Types** tab.

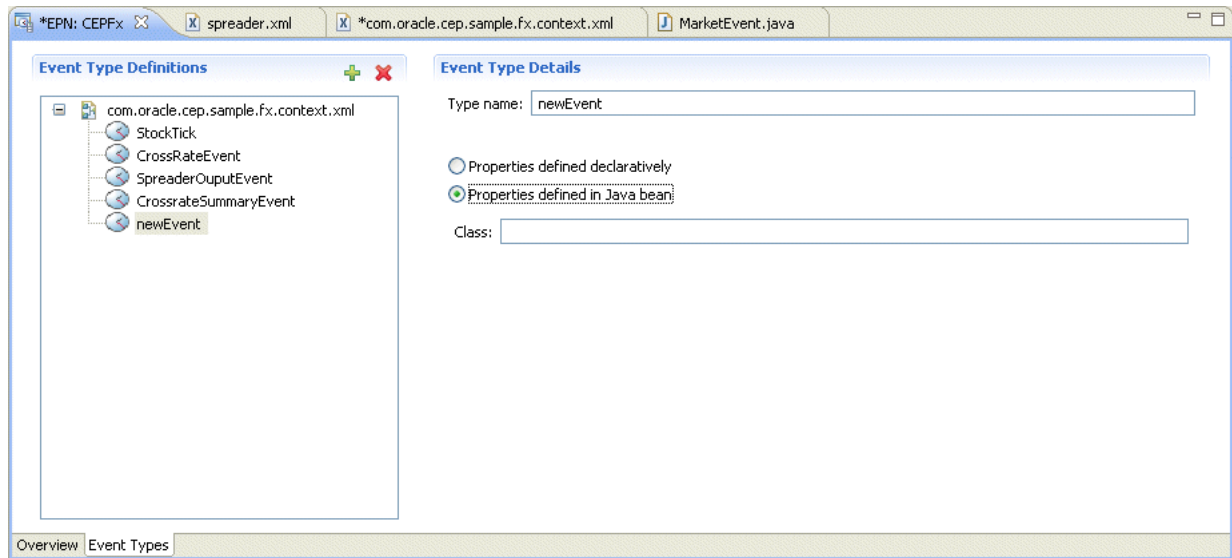
The Event Type tab appears as [Figure 2–2](#) shows.

Figure 2–2 Event Type Repository Editor



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 2–3](#) shows.

Figure 2–3 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 2–3](#) 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 Eclipse 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).
11. Use the event type:

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

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

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

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

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

2.2.2 How to Create an Oracle CEP Event Type as a JavaBean Manually

This procedure describes how to create and register an Oracle CEP event type as a JavaBean manually.

Alternatively, you can create and register your event type as a JavaBean using the Oracle CEP IDE for Eclipse event type repository editor (see [Section 2.2.1, "How to Create an Oracle CEP Event Type as a JavaBean Using the Event Type Repository Editor"](#)).

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

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

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

Oracle recommends that, if possible, you make your event type JavaBeans immutable to improve performance. For more information, see [Section 2.1.1, "Event Type Instantiation and Immutability"](#).

When you design your event, you must restrict your design to the even data types that [Section 2.1.3.1, "Event Types Specified as JavaBean or Java Class"](#) describes.

[Example 2–5](#) shows the `MarketEvent` which is implemented by the `com.bea.wlevs.example.algotrading.event.MarketEvent` class.

Example 2–5 MarketEvent Class

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

import java.util.Date;

public final class MarketEvent {
    private final Long timestamp;

    private final String symbol;

    private final Double price;

    private final Long volume;

    private final Long latencyTimestamp;

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

    public Double getPrice() {
        return price;
    }

    public String getSymbol() {
        return symbol;
    }
}
```

```

    }

    public Long getTimestamp() {
        return timestamp;
    }

    public Long getLatencyTimestamp() {
        return latencyTimestamp;
    }

    public Long getVolume() {
        return volume;
    }
}

```

2. Compile the JavaBean that represents your event type.
3. Register your JavaBean event type in the Oracle CEP event type repository:
 - a. To register declaratively, edit the EPN assembly file using the `wlevs:event-type-repository` element `wlevs:event-type` child element as [Example 2-6](#) shows.

Example 2-6 EPN Assembly File event-type-repository

```

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

```

- b. To register programatically, use the `EventTypeRepository` class as [Example 2-7](#) shows.

Example 2-7 Programmatically Registering an Event

```

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

```

For more information, see [Section 2.7, "Accessing the Event Type Repository"](#).

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

```

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

```

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

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

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

2.3 Creating an Oracle CEP Event Type as a Tuple

You can create and register an Oracle CEP event type as a tuple.

When you design your event, you must restrict your design to the even data types that [Section 2.1.3.3, "Event Types Specified as a Tuple"](#) describes.

This topic describes:

- [Section 2.3.1, "How to Create an Oracle CEP Event Type as a Tuple Using the Event Type Repository Editor"](#)
- [Section 2.3.2, "How to Create an Oracle CEP Event Type as a Tuple Manually"](#)

2.3.1 How to Create an Oracle CEP Event Type as a Tuple Using the Event Type Repository Editor

This procedure describes how to create and register an Oracle CEP event type as a tuple using the Oracle CEP IDE for Eclipse event type repository editor. For more information about the Oracle CEP IDE for Eclipse, see [Example 3, "Overview of the Oracle CEP IDE for Eclipse"](#).

Alternatively, you can create and register your event type as a tuple manually (see [Section 2.3.2, "How to Create an Oracle CEP Event Type as a Tuple Manually"](#)).

To create an Oracle CEP event type as a tuple using the event type repository editor:

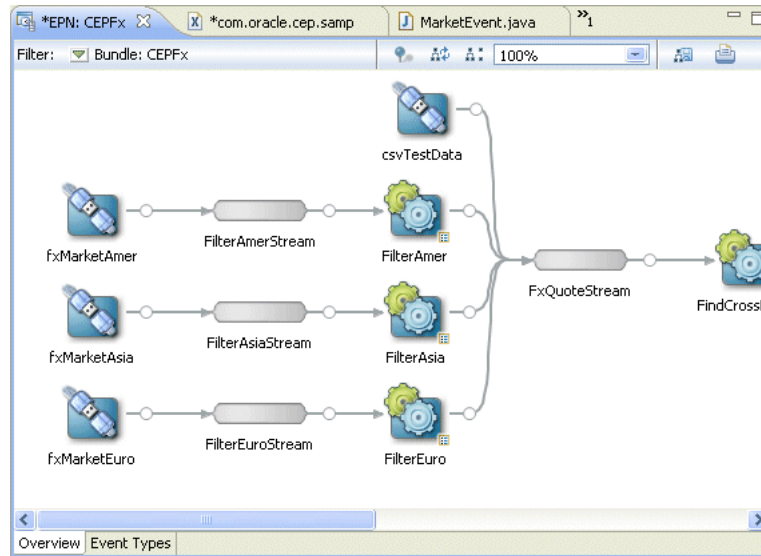
1. Decide on the properties your event type requires.

When you design your event, you must restrict your design to the even data types that [Section 2.1.3.3, "Event Types Specified as a Tuple"](#) describes.

2. Open the EPN in the Oracle CEP IDE for Eclipse.

The EPN editor opens as [Figure 2–4](#) shows.

Figure 2–4 EPN Editor

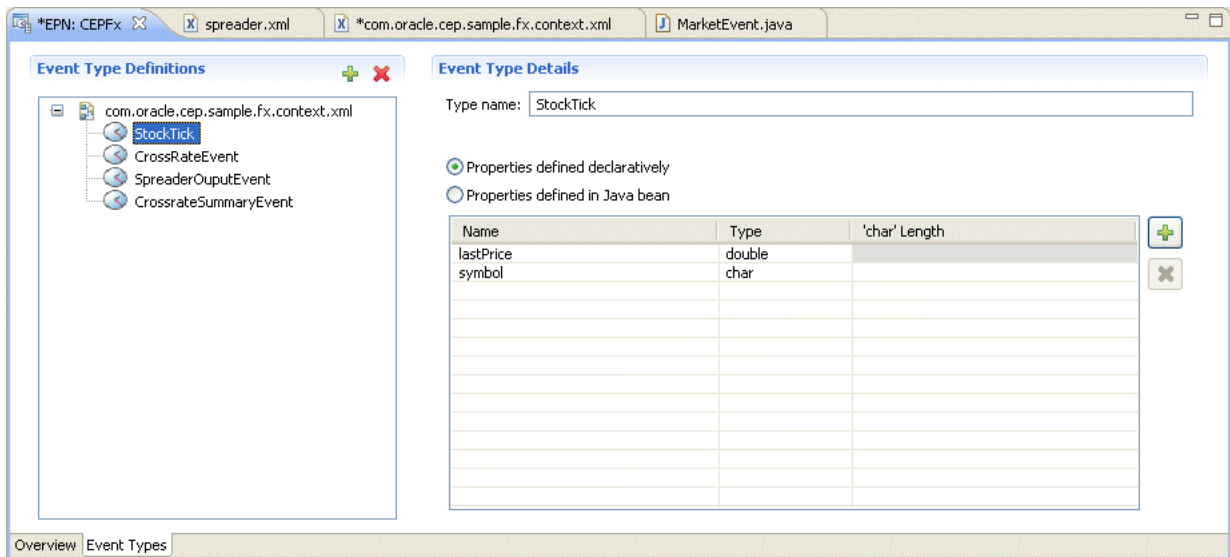


For more information, see [Section 6.1, "Opening the EPN Editor"](#).

3. Click the **Event Type** tab.

The Event Type tab appears as [Figure 2–5](#) shows.

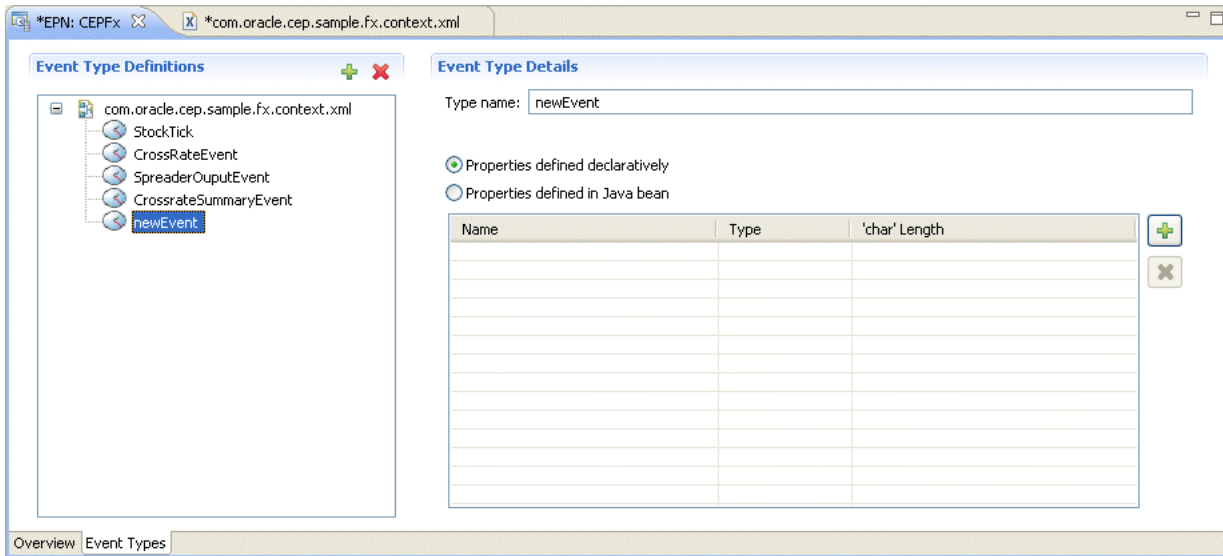
Figure 2–5 Event Type Repository Editor



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 2–6](#) shows.

Figure 2–6 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 2–6](#) 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 2.1.3.3, "Event Types Specified as a Tuple"](#) 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 Eclipse 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).
10. Use the event type:

- Reference the event types using the `EventTypeRepository` introspection interface `EventType` in the Java code of the adapters and business logic POJO in your application:

```

@Service
public void setEventTypeRepository(EventTypeRepository etr) {
    etr_ = etr;
}
...
// handle events
public void onInsertEvent(Object event) throws EventRejectedException {
    // 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();

    // test if property is present
    if(eventType.isProperty("fromRate")) {
        // get property value
        Object propValue =
eventType.getProperty("fromRate").getValue(event);
    }
    ...
}

```

For more information, see [Section 2.7, "Accessing the Event Type Repository"](#).

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

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>

```

2.3.2 How to Create an Oracle CEP Event Type as a Tuple Manually

This procedure describes how to create and register an Oracle CEP event type declaratively in the EPN assembly file as a tuple. Oracle recommends that you create an Oracle CEP event type as a JavaBean (see [Section 2.2.2, "How to Create an Oracle CEP Event Type as a JavaBean Manually"](#)).

For more information on valid event type data types, see [Section 2.1.3.3, "Event Types Specified as a Tuple"](#).

To create an Oracle CEP event type as a tuple:

1. Decide on the properties your event type requires.

When you design your event, you must restrict your design to the even data types that [Section 2.1.3.3, "Event Types Specified as a Tuple"](#) describes.

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

To register declaratively, edit the EPN assembly file using the `wlevs:event-type-repository` element `wlevs:event-type` child element as [Example 2-8](#) shows.

Example 2-8 EPN Assembly File event-type-repository

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

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

For more information on the valid values of the `type` attribute, see [Section 2.1.3.3, "Event Types Specified as a Tuple"](#).

3. Use the event type:
 - Reference the event types using the `EventTypeRepository` introspection interface `EventType` in the Java code of the adapters and business logic POJO in your application:

```
@Service
public void setEventTypeRepository(EventTypeRepository etr) {
    etr_ = etr;
}
...
// handle events
public void onInsertEvent(Object event) throws EventRejectedException {
    // 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();

    // test if property is present
    if(eventType.isProperty("fromRate")) {
        // get property value
        Object propValue =
            eventType.getProperty("fromRate").getValue(event);
    }
    ...
}
```

For more information, see [Section 2.7, "Accessing the Event Type Repository"](#).

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

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>
```

2.4 Creating an Oracle CEP Event Type as a Java Class

You can create and register an Oracle CEP event type as a Java class.

This topic describes:

- [Section 2.4.1, "How to Create an Oracle CEP Event Type as a Java Class Manually"](#)

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

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

For more information on valid event type data types, see [Section 2.1.3.1, "Event Types Specified as JavaBean or Java Class"](#).

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

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

Oracle recommends that, if possible, you make your event type Java class immutable to improve performance. For more information, see [Section 2.1.1, "Event Type Instantiation and Immutability"](#).

[Example 2–5](#) shows the `ForeignExchangeEvent` which is implemented by the `ForeignExchangeEvent` inner class of `com.bea.wlevs.example.fx.OutputBean`.

Example 2–9 ForeignExchangeEvent Class: Does not Conform to JavaBean Standards

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

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

    public ForeignExchangeEvent() {

    }
}
```

```

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

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

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

    public String getSymbol() {
        return symbol;
    }

    public Double getPrice() {
        return price;
    }

    public String getFromRate() {
        return fromRate;
    }

    public String getToRate() {
        return toRate;
    }

    public Long getClientTimestamp() {
        return clientTimestamp;
    }

    public Long getAdapterTimestamp() {
        return adapterTimestamp;
    }
}

```

2. Create a Java class to represent your event type builder as [Example 2–10](#) shows.

Example 2–10 ForeignExchangeBuilderFactory

```

package com.bea.wlevs.example.fx;

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

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

public class ForeignExchangeBuilderFactory implements EventBuilder.Factory {

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

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

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

```

```

        (String) values.get("toRate"));
    }

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

```

Oracle CEP uses the event builder factory to create event instances that do not follow the JavaBean specification. See [Section 2.6, "Using an Event Type Builder Factory"](#) for additional information about using an event type builder factory.

3. Compile the Java classes that represent your event type and event builder factory.
4. Register your event type and event builder factory in the Oracle CEP event type repository:

- a. To register declaratively, edit the EPN assembly file using the `wlevs:event-type-repository` element `wlevs:event-type` child element as [Example 2-6](#) shows.

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

Example 2-11 EPN Assembly File event-type-repository

```

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

```

- b. To register programatically, use the `EventTypeRepository` class as [Example 2-7](#) shows.

Example 2-12 Programmatically Registering an Event

```

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

```

For more information, see [Section 2.7, "Accessing the Event Type Repository"](#).

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

```

public void onEvent(List newEvents)
    throws RejectEventException {
    for (Object event : newEvents) {
        ForeignExchangeEvent fxEvent = (ForeignExchangeEvent) event;

```

```

        System.out.println("Price: " + fxEvent.getPrice());
    }
}

```

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

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

```

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

```

2.5 Creating an Oracle CEP Event Type as a `java.util.Map`

You can create and register an Oracle CEP event type as a `java.util.Map`.

This topic describes:

- [Section 2.5.1, "How to Create an Oracle CEP Event Type as a `java.util.Map`"](#)

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

This procedure describes how to create and register an Oracle CEP event type as a `java.util.Map`. Oracle recommends that you create an Oracle CEP event type as a `JavaBean` (see [Section 2.2.2, "How to Create an Oracle CEP Event Type as a `JavaBean` Manually"](#)).

For more information on valid event type data types, see [Section 2.1.3.2, "Event Types Specified as `java.util.Map`"](#).

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

1. Decide on the properties your event type requires.
2. Register your event type in the Oracle CEP event type repository:
 - a. To register declaratively, edit the EPN assembly file using the `wlevs:event-type-repository` element `wlevs:event-type` child element as [Example 2–13](#) shows.

Example 2–13 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 CEP 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 2–14](#) shows.

Example 2–14 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 2.7, "Accessing the Event Type Repository"](#).

3. Use the event type:

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

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

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

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

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

2.6 Using an Event Type Builder Factory

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

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

When you program the event type builder factory, you must implement the `EventBuilder.Factory` inner interface of the `com.bea.wlevs.ede.api.EventBuilder` interface; see the *Oracle Fusion Middleware Java API Reference for Oracle Complex 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 {
    public EventBuilder createBuilder() {
        return new ForeignExchangeBuilder();
    }
}
```

```

    }
    static class ForeignExchangeBuilder implements EventBuilder {
        private Map<String, Object> values = new HashMap<String, Object>(10);
        public Object createEvent() {
            return new ForeignExchangeEvent(
                (String) values.get("symbol"),
                (Double) values.get("price"),
                (String) values.get("fromRate"),
                (String) values.get("toRate"));
        }
        public void put(String property, Object value) throws IllegalStateException {
            values.put(property, value);
        }
    }
}

```

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

```

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

```

2.7 Accessing the Event Type Repository

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 2.7.1, "Using the EPN Assembly File"](#)
- [Section 2.7.2, "Using the Spring-DM @ServiceReference Annotation"](#)
- [Section 2.7.3, "Using the Oracle CEP @Service Annotation"](#)

For more information, see *Oracle Fusion Middleware Java API Reference for Oracle Complex Event Processing*.

2.7.1 Using the EPN Assembly File

You can access the `EventTypeRepository` by specifying an `osgi:reference` in the EPN assembly file as [Example 2–15](#) shows.

Example 2–15 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 2–16](#) shows.

Example 2–16 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);
        ...
    }
}
```

2.7.2 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 2–17](#) shows.

Example 2–17 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) {
    ...
}
```

2.7.3 Using the Oracle CEP @Service Annotation

You can access the `EventTypeRepository` by using the Oracle CEP `@Service` annotation to initialize a property in your Java source as [Example 2–17](#) shows.

Example 2–18 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 I.5, "com.bea.wlevs.util.Service"](#).

2.8 Sharing Event Types Between Application Bundles

Each Oracle CEP application gets its own Java classloader and loads application classes using that classloader. This means that, by default, one application cannot

access the classes in another application. However, because the event type repository is a singleton service, you can configure the repository in one bundle and then explicitly export the event type classes so that applications in separate bundles (deployed to the same Oracle CEP server) can use these shared event types.

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

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

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

For more information, see:

- [Section 2.1.3.1, "Event Types Specified as JavaBean or Java Class"](#)
- [Section 24.2.3, "Assembling Applications With Foreign Stages"](#)
- "Oracle Java Data Cartridge" in the *Oracle Complex Event Processing CQL Language Reference*

Part II

Oracle CEP IDE for Eclipse

Part II contains the following chapters:

- [Chapter 3, "Overview of the Oracle CEP IDE for Eclipse"](#)
- [Chapter 4, "Oracle CEP IDE for Eclipse Projects"](#)
- [Chapter 5, "Oracle CEP IDE for Eclipse and Oracle CEP Servers"](#)
- [Chapter 6, "Oracle CEP IDE for Eclipse and the Event Processing Network"](#)

Overview of the Oracle CEP IDE for Eclipse

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

This chapter describes:

- [Section 3.1, "Overview of Oracle CEP IDE for Eclipse"](#)
- [Section 3.2, "Installing the Latest Oracle CEP IDE for Eclipse"](#)
- [Section 3.3, "Installing the Oracle CEP IDE for Eclipse Distributed With Oracle CEP"](#)
- [Section 3.4, "Configuring Eclipse"](#)

3.1 Overview of Oracle CEP IDE for Eclipse

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

This section describes:

- [Section 3.1.1, "Features"](#)
- [Section 3.1.2, "JDK Requirements"](#)
- [Section 3.1.3, "Default Oracle CEP Domain ocep_domain and Development"](#)

For more information about Oracle CEP IDE for Eclipse, see <http://www.oracle.com/technology/products/event-driven-architecture/cep-ide/11/>.

3.1.1 Features

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

- Project creation wizards and templates to quickly get started building event driven applications.
- Advanced editors for source files including Java and XML files common to Oracle CEP applications.
- Integrated server management to seamlessly start, stop, and deploy to Oracle CEP server instances all from within the IDE.
- Integrated debugging.
- Event Processing Network (EPN) visual design views for orienting and navigating in event processing applications and visually creating and editing EPN components.

- Oracle CEP application source file validation including Oracle Continuous Query Language (Oracle CQL) syntax highlighting and component configuration and assembly files.
- Ability to build and export deployable Oracle CEP applications.
- Integrated support for the Oracle CEP Visualizer so you can use the Oracle CEP Visualizer from within the IDE.

3.1.2 JDK Requirements

In 11g Release 1 (11.1.1), Oracle CEP IDE for Eclipse requires JDK 6.0. For more information, see:

- "Setting Your Development Environment" in the *Oracle Complex Event Processing Getting Started*
- [Section 3.4, "Configuring Eclipse"](#)

3.1.3 Default Oracle CEP Domain ocep_domain and Development

When you choose a **Typical** Oracle CEP 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 CEP 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 CEP IDE for Eclipse during development.

If you choose a **Typical** Oracle CEP server install, you can use the Configuration Wizard to create an Oracle CEP server domain.

For more information, see:

- "Installation Overview" in the *Oracle Complex Event Processing Getting Started*
- "Creating an Oracle CEP Standalone-Server Domain" in the *Oracle Complex Event Processing Administrator's Guide*
- "Creating an Oracle CEP Multi-Server Domain Using Oracle CEP Native Clustering" in the *Oracle Complex Event Processing Administrator's Guide*

3.2 Installing the Latest Oracle CEP IDE for Eclipse

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

To install the latest Oracle CEP IDE for Eclipse:

1. Obtain the required versions of Eclipse (3.5.1) and WTP (2.0). We recommend you take the entire Galileo installation available at the following Web sites:

Windows:

<http://www.eclipse.org/downloads/download.php?file=/technology/epp/downloads/release/galileo/SR1/eclipse-jee-galileo-SR1-win32.zip>

Linux:

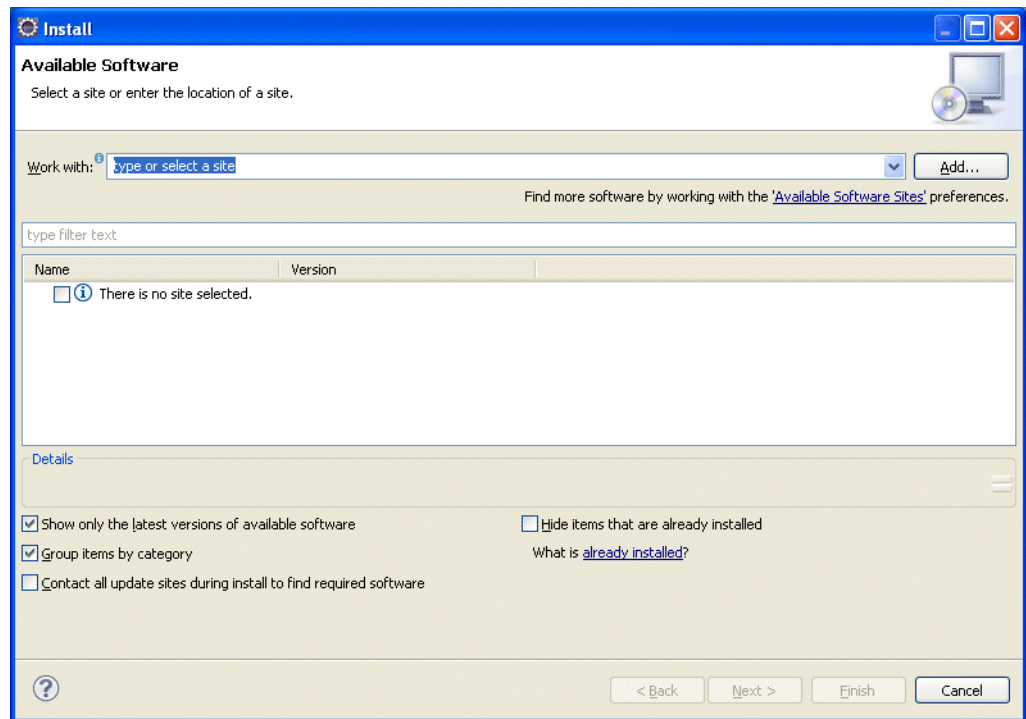
<http://www.eclipse.org/downloads/download.php?file=/technology/epp/downloads/release/galileo/SR1/eclipse-jee-galileo-SR1-linux.tar.gz>

[y/epp/downloads/release/galileo/SR1/eclipse-jee-galileo-SR1-1
linux-gtk.tar.gz](http://www.oracle.com/technologies/soa/complex-event-processing.html)

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

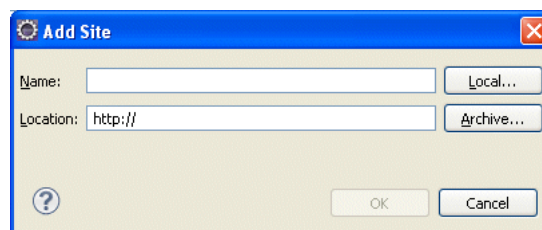
- Open your Eclipse IDE and select the menu item **Help > Install New Software**. The Install dialog appears as [Figure 3–1](#) shows.

Figure 3–1 Install Dialog



- Click **Add**. The Add Site dialog appears as [Figure 3–2](#) shows.

Figure 3–2 Add Site Dialog



- Configure this dialog as [Table 3–1](#) describes.

Table 3–1 New Update Site Dialog Attributes

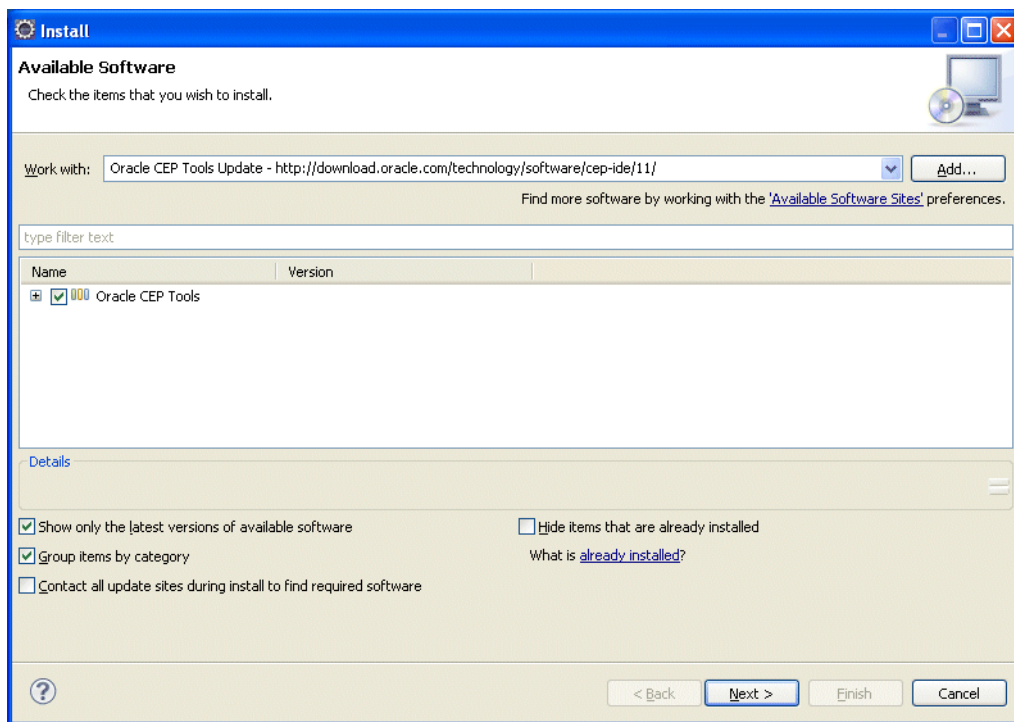
Attribute	Description
Name	The name for this remote update site. For example: Oracle CEP Tools Update.
URL	The URL to the remote update site. Valid value: http://download.oracle.com/technology/software/cep-ide/11/

- Click **OK**.
- In the Install dialog, from the **Work with** pull down menu, select the Oracle CEP Tools Update site you just created.

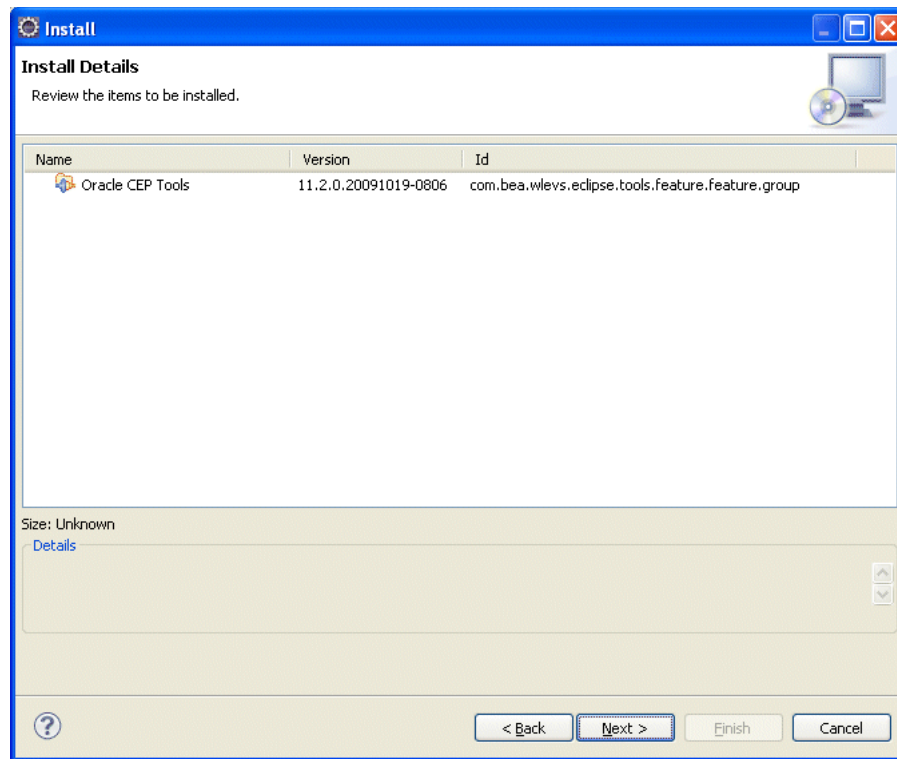
It may take a few moments for Eclipse to contact the remote update site. During this time, the "There is no site selected" entry reads "Pending".

When Eclipse has made contact with the remote update site, the Oracle CEP Tools entry appears in the list of update sites as [Figure 3–3](#) shows.

- Check the check box next to the **Oracle CEP Tools** entry as [Figure 3–3](#) shows.

Figure 3–3 Install Dialog - Site Selected

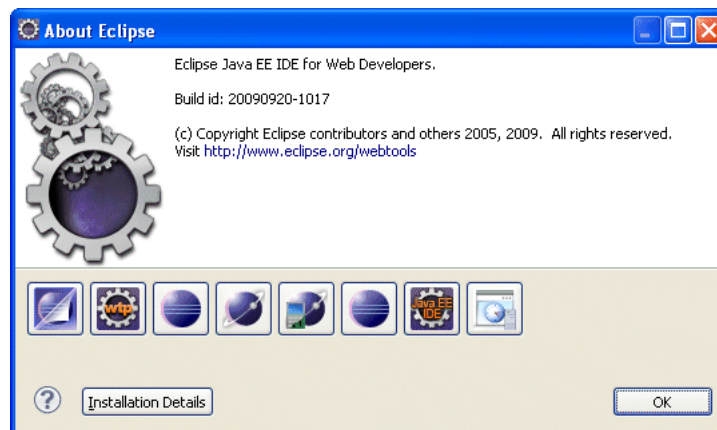
- Click **Next**.
The Install Details dialog appears as [Figure 3–4](#) shows.

Figure 3–4 Install Dialog - Install Details**9. Click Next.**

The Review Licenses dialog appears.

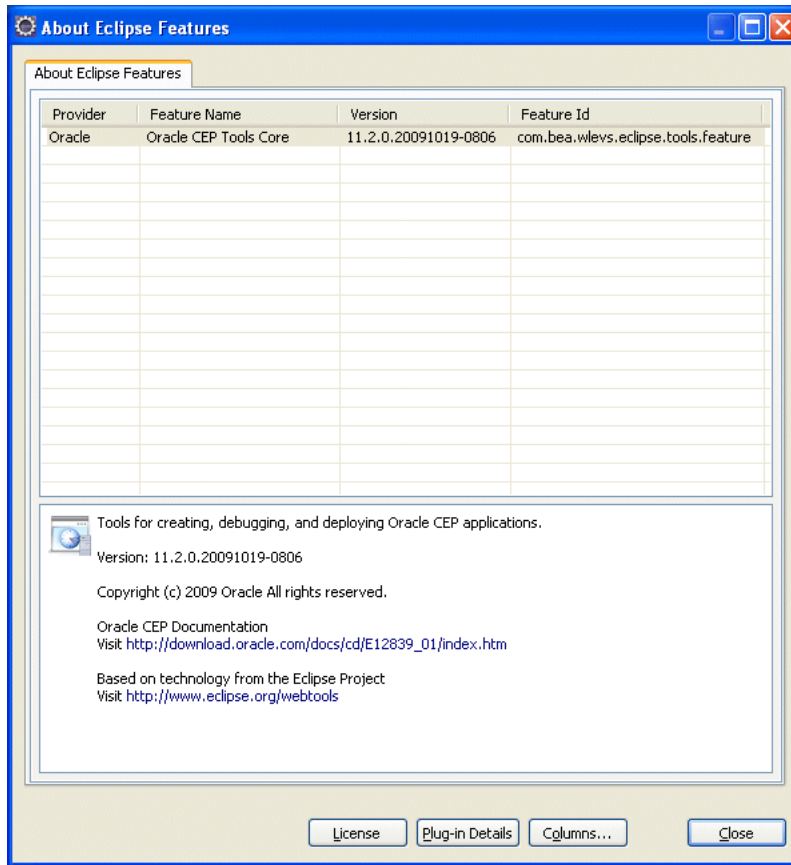
10. Click Finish.**11. When prompted restart, Eclipse. If you skip this, unreliable behavior can occur.****12. To confirm the installation, select Help > About Eclipse.**

The About Eclipse dialog appears as [Figure 3–5](#) shows.

Figure 3–5 About Eclipse**13. Click Oracle.**

The About Eclipse Features dialog appears as [Figure 3–6](#) shows.

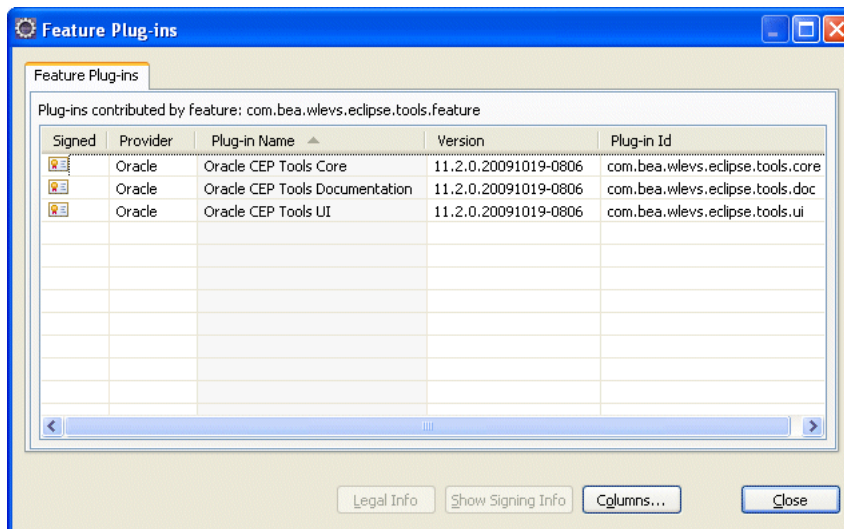
Figure 3–6 About Eclipse Features Dialog



14. Click Plug-In Details.

The Feature Plug-ins dialog appears as [Figure 3–7](#) shows.

Figure 3–7 Feature Plug-ins Dialog



15. Confirm that the plug-ins that [Table 3–2](#) lists are shown.

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

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

16. After installing Oracle CEP IDE for Eclipse, consider the following topics:
- [Section 3.1.3, "Default Oracle CEP Domain ocep_domain and Development"](#)
 - [Section 3.4, "Configuring Eclipse"](#)
 - [Appendix J, "Oracle CEP IDE for Eclipse Tutorial"](#)

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

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

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

1. Obtain the required versions of Eclipse (3.5.1) and WTP (2.0). We recommend you take the entire Galileo installation available at the following Web sites:

Windows:

<http://www.eclipse.org/downloads/download.php?file=/technology/epp/downloads/release/galileo/SR1/eclipse-jee-galileo-SR1-win32.zip>

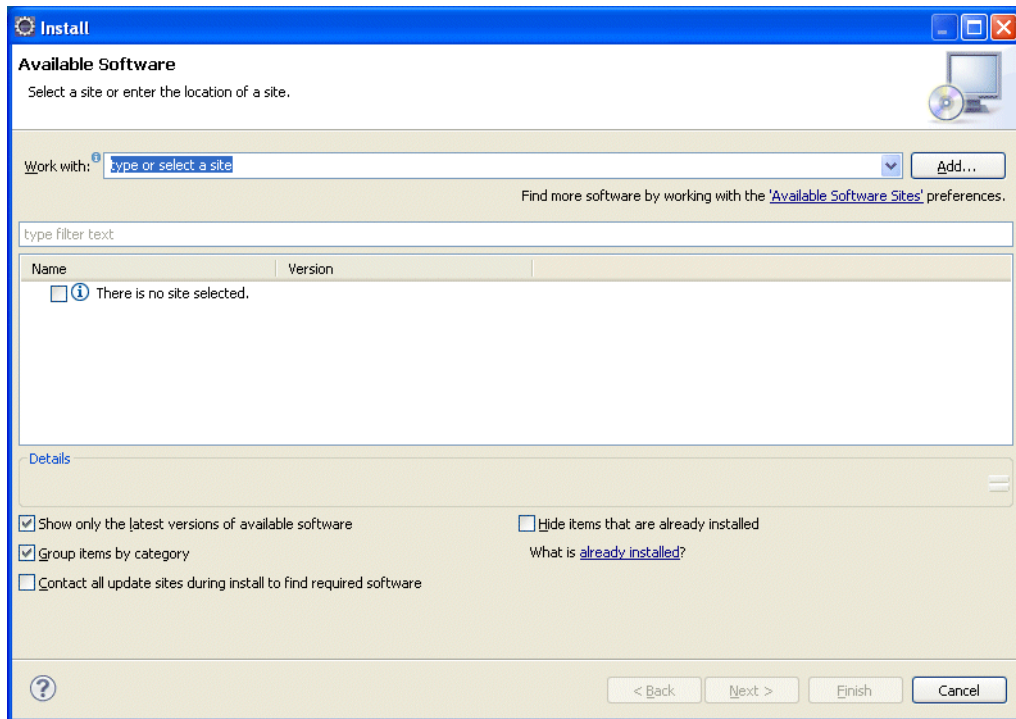
Linux:

<http://www.eclipse.org/downloads/download.php?file=/technology/epp/downloads/release/galileo/SR1/eclipse-jee-galileo-SR1-linux-gtk.tar.gz>

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

2. Open your Eclipse IDE and select the menu item **Help > Install New Software**. The Install dialog appears as [Figure 3–1](#) shows.

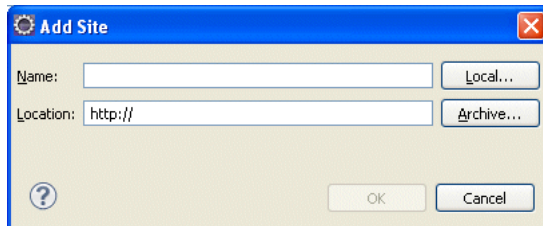
Figure 3–8 Install Dialog



3. Click **Add**.

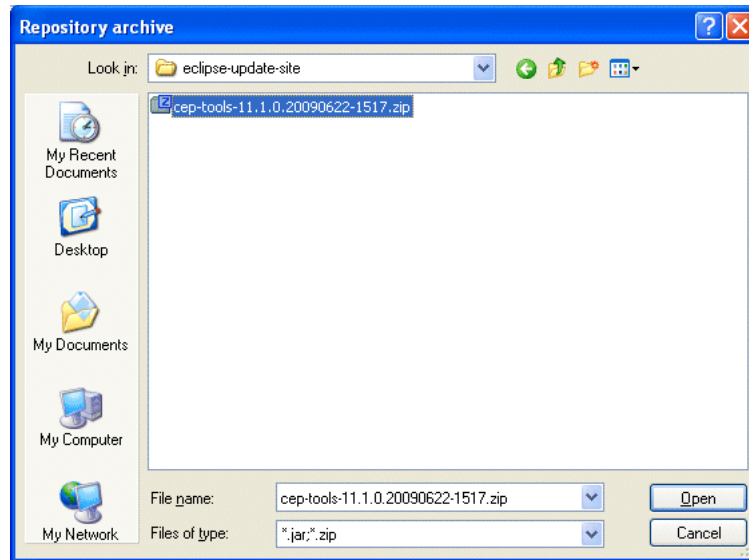
The Add Site dialog appears as [Figure 3–2](#) shows.

Figure 3–9 Add Site Dialog



4. Click **Archive**.

The Select Local Site Archive dialog appears as [Figure 3–10](#) shows.

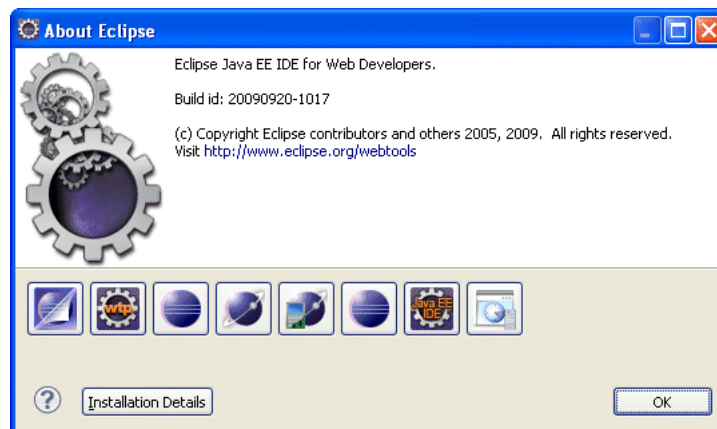
Figure 3–10 Select Local Site Archive Dialog

5. Navigate to the `ORACLE_CEP_HOME/ocp_11.1/eclipse-update-site` directory and select the `cep-tools-11.1.0.DATE-BUILD.zip` file.

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

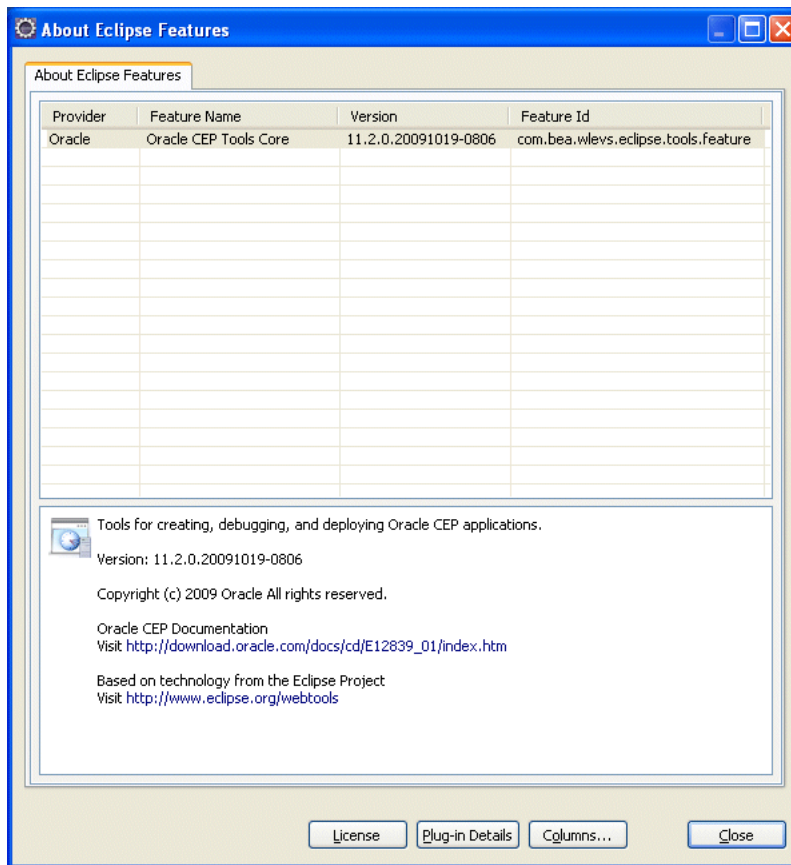
6. Click **Open**.
7. Complete the Update Manager, selecting to install the Oracle CEP tools.
8. When prompted restart, Eclipse. If you skip this, unreliable behavior can occur.
9. To confirm the installation, select **Help > About Eclipse**.

The About Eclipse dialog appears as [Figure 3–5](#) shows.

Figure 3–11 About Eclipse

10. Click **Oracle**.

The About Eclipse Features dialog appears as [Figure 3–6](#) shows.

Figure 3–12 About Eclipse Features Dialog**11. Click Plug-In Details.**

The Feature Plug-ins dialog appears as [Figure 3–7](#) shows.

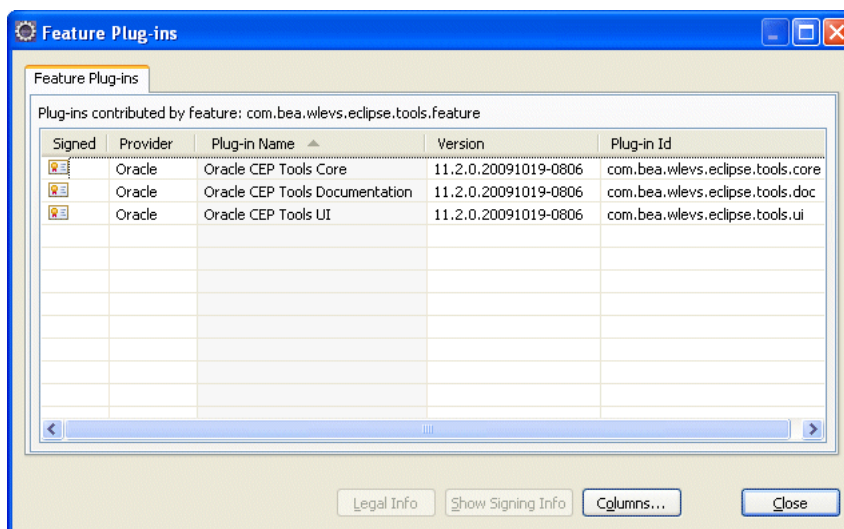
Figure 3–13 Feature Plug-ins Dialog**12. Confirm that the plug-ins that [Table 3–2](#) lists are shown.**

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

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

13. After installing Oracle CEP IDE for Eclipse, consider the following topics:
- [Section 3.1.3, "Default Oracle CEP Domain ocep_domain and Development"](#)
 - [Section 3.4, "Configuring Eclipse"](#)
 - [Appendix J, "Oracle CEP IDE for Eclipse Tutorial"](#)

3.4 Configuring Eclipse

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

To configure Eclipse:

1. Exit out of Eclipse if it is running.
2. Install a Java 6 JRE on your computer.
By default, the JRockit Java 6.0 JRE is installed with Oracle CEP in your Oracle CEP home directory. For example:
C:\OracleCEP\jrockit_160_20\jre
3. Using the editor of your choice, open your `eclipse.ini` file located in your Eclipse install directory, for example, C:\eclipse\ as [Example 3–1](#) shows.

Example 3–1 Default eclipse.ini File

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

Note: When making changes to the `eclipse.ini` file, add arguments one argument per line, as <http://wiki.eclipse.org/Eclipse.ini> describes.

For more information about configuring Eclipse, see http://wiki.eclipse.org/FAQ_How_do_I_run_Eclipse.

4. Add the following lines to the `eclipse.ini` file as [Example 3–2](#) shows.

Example 3–2 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 3-3](#) shows.

Example 3-3 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:

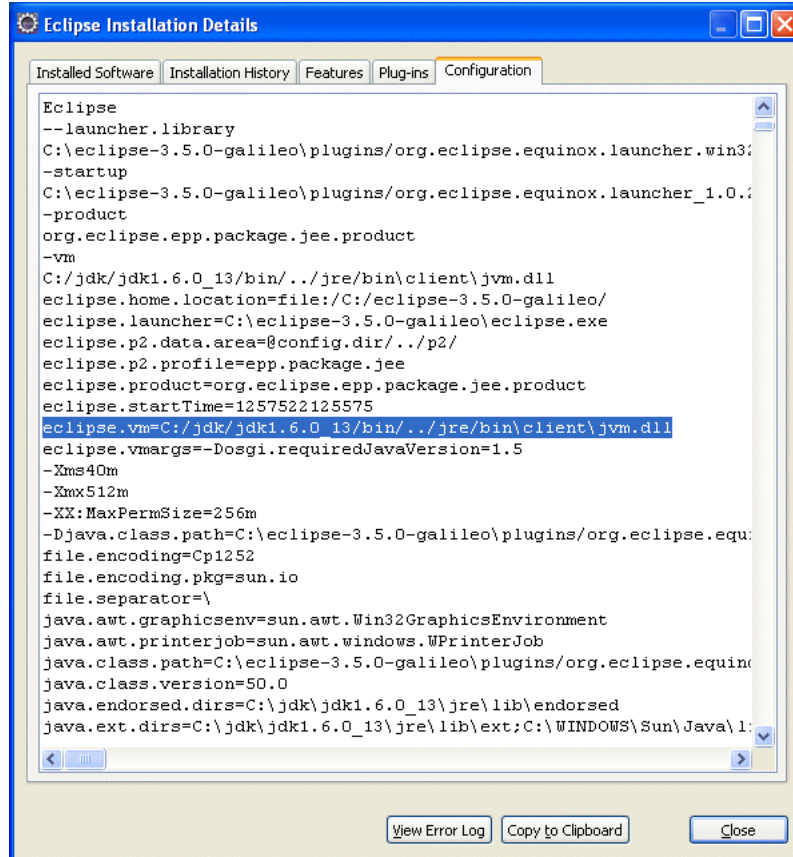
```
-vm
C:\OracleCEP\jrockit_160_20\jre\bin\javaw.exe
```

Note: Do not put both the `-vm` and the path on the same line. Each must be on a separate line as [Example 3-3](#) 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 3-14](#).

Figure 3-14 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 CEP IDE for Eclipse Projects

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

This chapter describes:

- [Section 4.1, "Oracle CEP Project Overview"](#)
- [Section 4.2, "Creating Oracle CEP Projects"](#)
- [Section 4.3, "Creating EPN Assembly Files"](#)
- [Section 4.4, "Creating Component Configuration Files"](#)
- [Section 4.5, "Exporting Oracle CEP Projects"](#)
- [Section 4.6, "Upgrading Projects"](#)
- [Section 4.7, "Managing Libraries and Other Non-Class Files in Oracle CEP Projects"](#)
- [Section 4.8, "Configuring Oracle CEP IDE for Eclipse Preferences"](#)

4.1 Oracle CEP Project Overview

An Oracle CEP application includes the following artifacts:

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

[Figure 4–1](#) shows the Explorer after creating a project.

Figure 4–1 Oracle CEP Project Structure

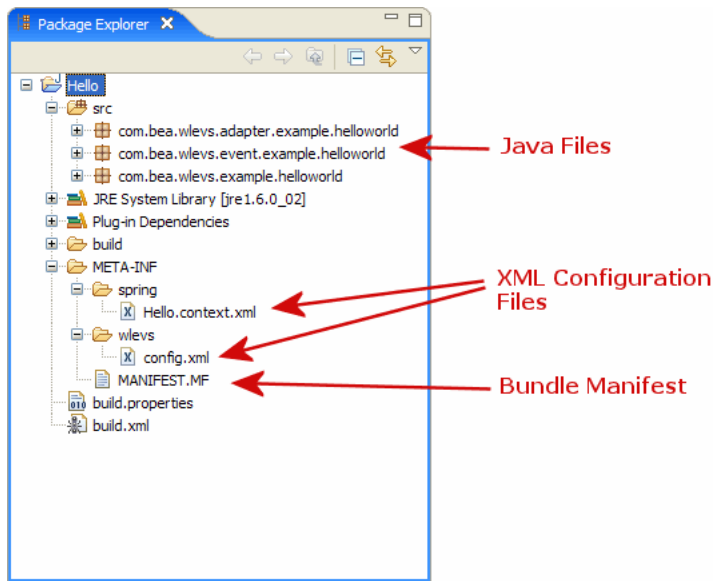


Table 4–1 summarizes the important files in an Oracle CEP project including their use and location.

Table 4–1 Oracle CEP 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 CEP application with Java files. All Java files must be in a source folder in order to be compiled. For more information, see Chapter 1, "Overview of Creating Oracle CEP Applications" .
EPN assembly file	<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 Chapter 1.1.4, "EPN Assembly File" .
Processor configuration file	<code>META-INF/wlevs</code>	The processor configuration file is where the Complex Event Processor (CEP) is defined. In this file you'll find processor rules (defined in the Continuous Query Language - CQL or the Event Processing Language--EPL) and other component configuration settings. For more information, see: <ul style="list-style-type: none"> ▪ Chapter 10, "Configuring Oracle CQL Processors" ▪ Chapter 11, "Configuring EPL Processors"
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 Chapter 24, "Assembling and Deploying Oracle CEP Applications" .

4.2 Creating Oracle CEP Projects

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

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

4.2.1 How to Create an Oracle CEP Project

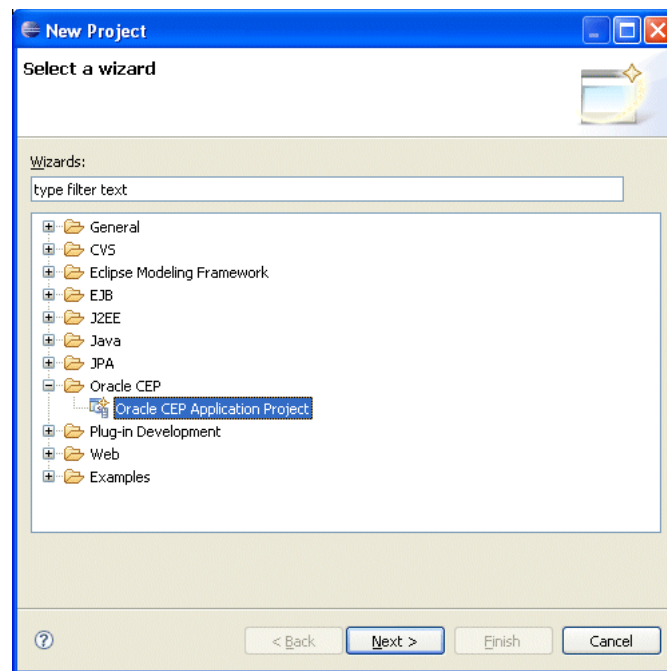
By default new projects are set to use Java 6.0. This section describes how to create an Oracle CEP project using Java 6. For information on configuring an Oracle CEP project to use Java 6, see [Section 3.4, "Configuring Eclipse"](#).

To create an Oracle CEP project:

1. Open the EPN Editor (see [Section 6.1, "Opening the EPN Editor"](#))
2. Select **File > New Project**.

The New Project - Select a Wizard dialog appears as shown in [Figure 4-2](#).

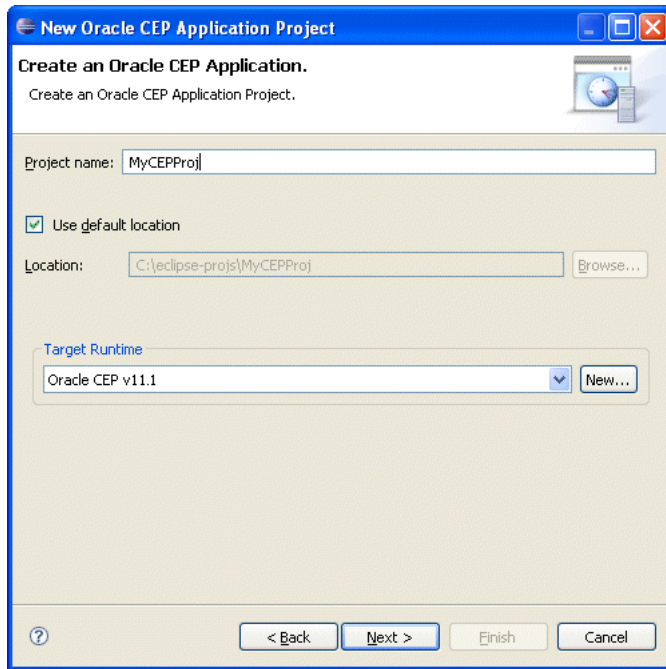
Figure 4-2 New Project - Select a Wizard Dialog



3. Expand **Oracle CEP** and select **Oracle CEP Application Project**.
4. Click **Next**.

The New Oracle CEP Application Project wizard appears as shown in [Figure 4-3](#).

Figure 4–3 New Oracle CEP Application Project Wizard: Create an Oracle CEP Application



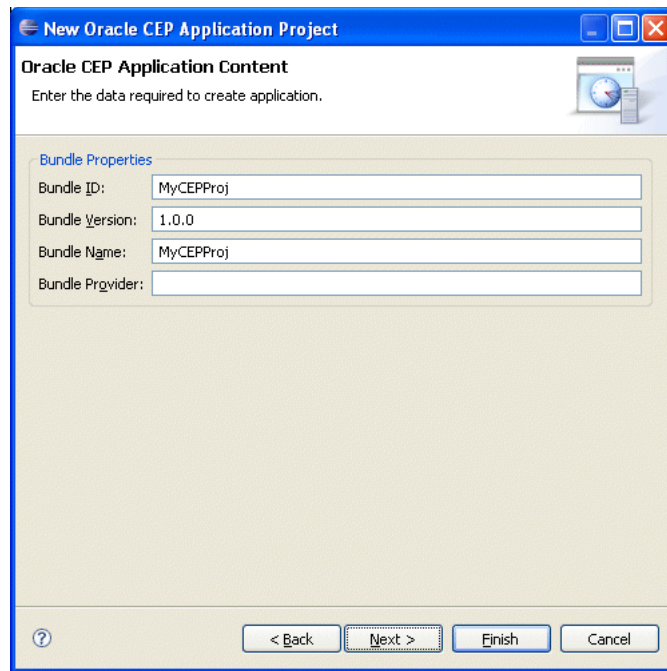
5. Configure the Create an Oracle CEP Application dialog as shown in [Table 4–2](#).

Table 4–2 Create an Oracle CEP Application Dialog

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

6. Click **Next**.

The Oracle CEP Application Content dialog appears as shown in [Figure 4–4](#).

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

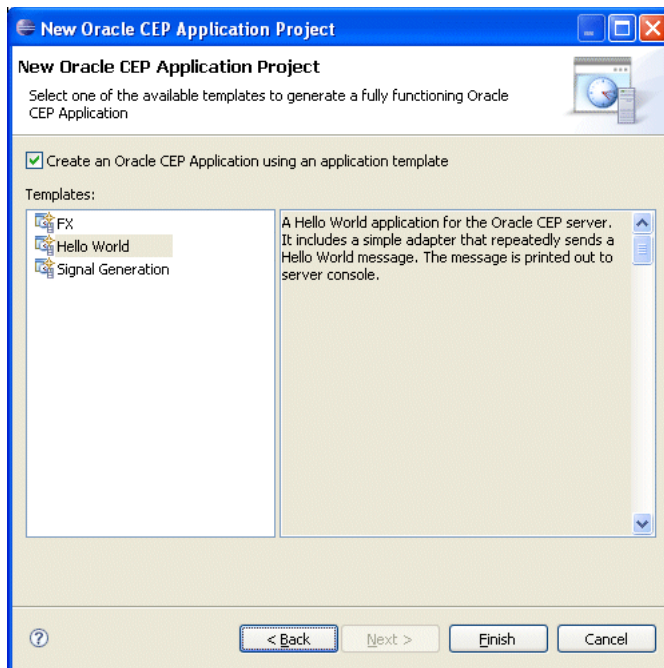
- Optionally, configure the Oracle CEP Application Content dialog as shown in [Table 4–3](#).

Table 4–3 Oracle CEP Application Content Dialog

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

- Click **Next**.

The Template dialog appears as shown in [Figure 4–5](#).

Figure 4–5 New Oracle CEP Application Project Wizard: Template Dialog

9. Optionally, select an Oracle CEP application template to pre-populate your project with the content that the template specifies.

10. Click **Finish**.

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

11. Optionally, create additional EPN assembly files and component configuration files.

By default, Oracle CEP 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 CEP objects in multiple EPN assembly and component configuration files to improve management, concurrent development, and re-use.

For more information, see:

- [Section 4.3, "Creating EPN Assembly Files"](#)
- [Section 4.4, "Creating Component Configuration Files"](#)

4.3 Creating EPN Assembly Files

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

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

4.3.1 How to Create a New EPN Assembly File Using Oracle CEP IDE for Eclipse

The most efficient and least error-prone way to create and edit the EPN file is using the New File Wizard in the Oracle CEP IDE for Eclipse.

For more information, see:

- [Section 6.2, "EPN Editor Overview"](#)
- [Section 4.7, "Managing Libraries and Other Non-Class Files in Oracle CEP Projects"](#)

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

1. Create an Oracle CEP project.

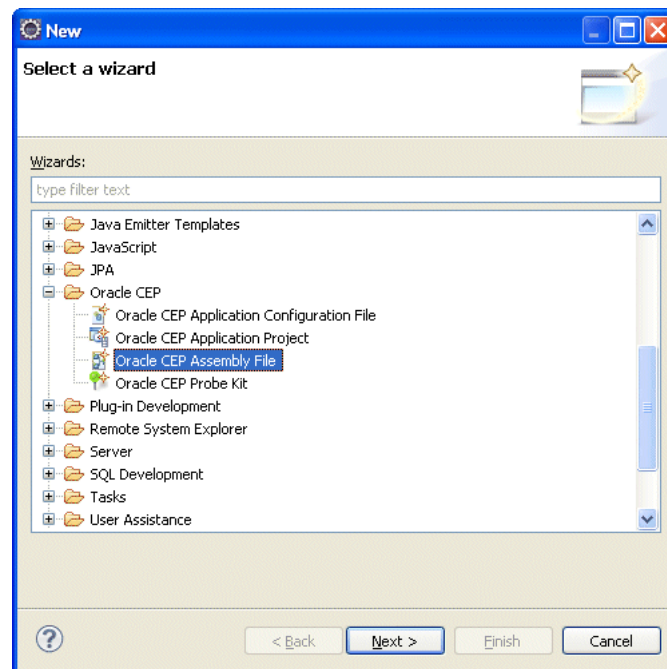
See [Section 4.2, "Creating Oracle CEP Projects"](#).

By default, Oracle CEP IDE for Eclipse creates one EPN assembly file for your project. Optionally, you may choose to define Oracle CEP 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 4–6](#) shows.

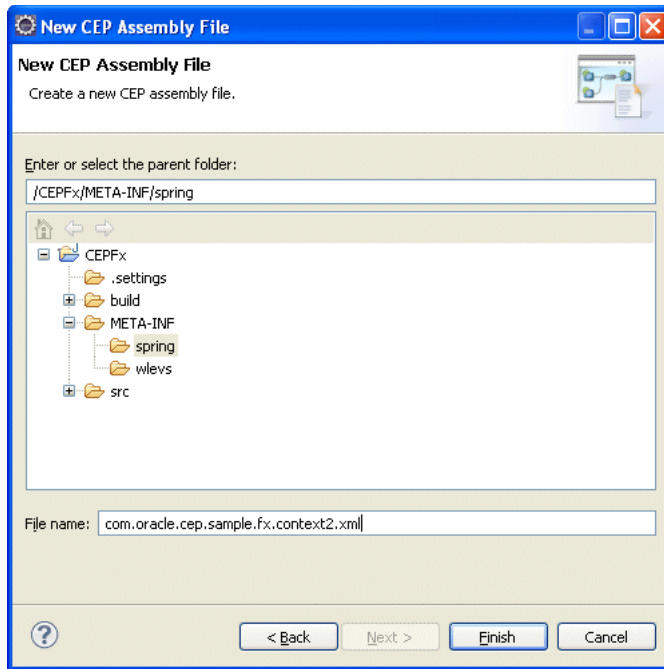
Figure 4–6 New Dialog



- b. Expand **Oracle CEP** and select **Oracle CEP Assembly File**.
 - c. Click **Next**.

The New CEP Assembly File dialog appears as [Figure 4–7](#) shows.

Figure 4–7 New CEP Assembly File Dialog



- d. Configure the New CEP Assembly File dialog as shown in [Table 4–4](#).

Table 4–4 New CEP Assembly File Dialog

Attribute	Description
Enter or select the parent folder	Enter the fully qualified path to the folder in which Oracle CEP 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 6.1, "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 6.2.2, "Filtering"](#).
 5. Create and connect nodes on the EPN.
See [Section 6.4, "Using the EPN Editor"](#).

4.4 Creating Component Configuration Files

You use a component configuration file to configure the components that make up your Oracle CEP application.

For an example of a component configuration file, see the "Foreign Exchange (FX) Example" in the *Oracle Complex Event Processing Getting Started*.

4.4.1 How to Create a New Component Configuration File Using Oracle CEP 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 CEP IDE for Eclipse.

For more information, see:

- [Section 6.2, "EPN Editor Overview"](#)
- [Section 4.7, "Managing Libraries and Other Non-Class Files in Oracle CEP Projects"](#)

To create a new component configuration file using Oracle CEP IDE for Eclipse:

1. Create an Oracle CEP project.

See [Section 4.2, "Creating Oracle CEP Projects"](#).

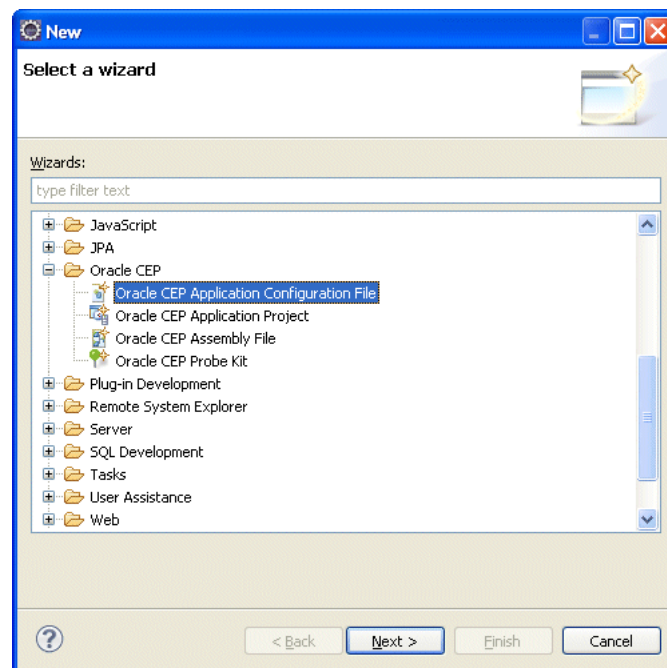
By default, Oracle CEP IDE for Eclipse creates one component configuration file for your project. Optionally, you may choose to configure Oracle CEP objects in multiple component configuration files to improve management, concurrent development, and re-use.

2. To optionally create component configuration files:

- Select **File > New > Other**.

The New dialog appears as [Figure 4-8](#) shows.

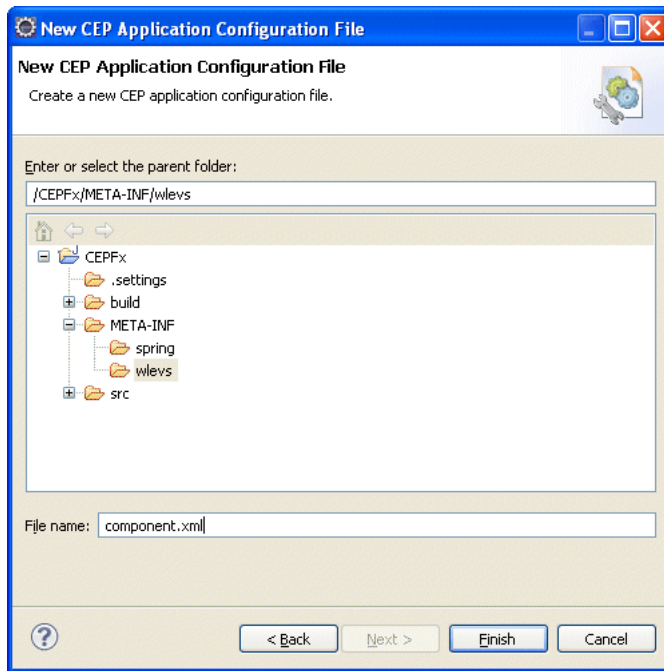
Figure 4-8 New Dialog



- Expand **Oracle CEP** and select **Oracle CEP Application Configuration File**.
- Click **Next**.

The New CEP Application Configuration File dialog appears as [Figure 4–9](#) shows.

Figure 4–9 *New CEP Application Configuration File Dialog*



- Configure the New CEP Assembly File dialog as shown in [Table 4–4](#).

Table 4–5 *New CEP Configuration File Dialog*

Attribute	Description
Enter or select the parent folder	Enter the fully qualified path to the folder in which Oracle CEP 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 6.1, "Opening the EPN Editor"](#).
 4. Create and connect nodes on the EPN.
See [Section 6.4, "Using the EPN Editor"](#).

4.5 Exporting Oracle CEP Projects

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

4.5.1 How to Export an Oracle CEP Project

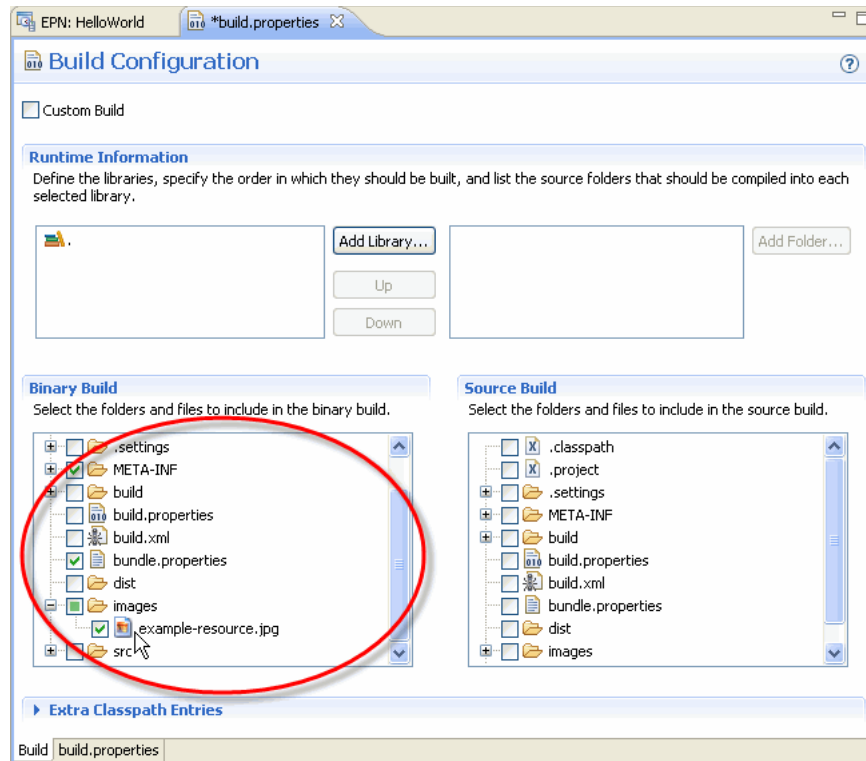
This section describes how to export an Oracle CEP project into an OSGi bundle.

To export an Oracle CEP project:

1. Start the Oracle CEP IDE for Eclipse and open your Oracle CEP project.
2. The Oracle CEP IDE for Eclipse compiles and adds Java resources to the exported JAR automatically. If your project contains other resources (such as a manifest file or images), configure your project to export them:
 - a. Locate the `build.properties` file in the Project Explorer and double-click this file to edit it.

The `build.properties` file opens as shown in [Figure 4–10](#).

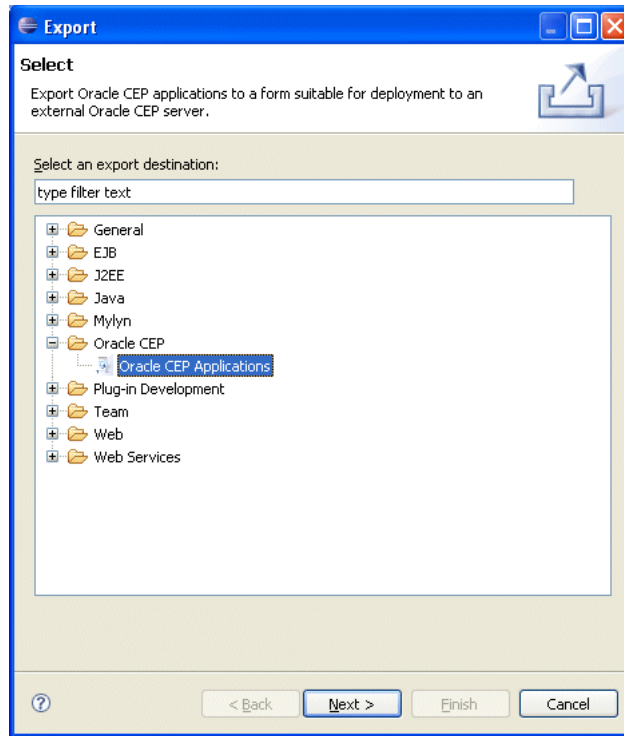
Figure 4–10 Oracle CEP Project `build.properties` File



- b. In the **Binary Build** area, check the resources you want exported with your application.
3. Select **File > Export**.

The Export dialog appears as shown in [Figure 4–11](#).

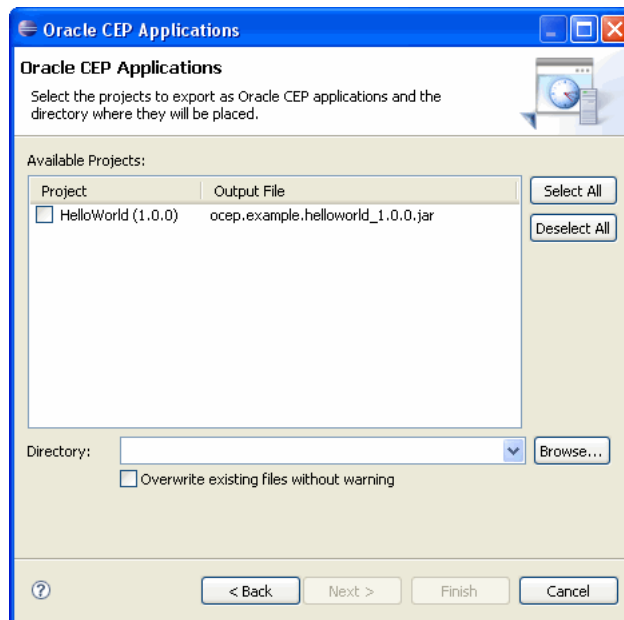
Figure 4–11 Export Dialog



4. Expand the **Oracle CEP** option and select **Oracle CEP Applications**.
5. Click **Next**.

The Oracle CEP Applications Export: Select Project dialog appears as shown in [Figure 4–12](#).

Figure 4–12 Oracle CEP Applications Export: Select Project Dialog



6. Configure the Oracle CEP Applications Export: Select Project dialog as shown in [Table 4–6](#).

Table 4–6 Oracle CEP Application Content Dialog

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

7. Click **Finish**.

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

8. Deploy the JAR file to your Oracle CEP server.
 - a. If your JAR is an application, see [Section 5.3.6, "How to Deploy an Application to an Oracle CEP Server"](#).
 - b. If your JAR is an application library, see [Section 24.1.3, "Application Libraries"](#)
9. Deploy other dependent resources, if any, to your Oracle CEP server.

For example:

- Other OSGi bundles your application depends on.
Deploy these bundles on the Oracle CEP server using the Oracle CEP Visualizer or command line deployment tools.
- Any entries in `config.xml` for datasources that are referenced from within the application.
Add these entries to the target server's `config.xml` file.

4.6 Upgrading Projects

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

This section describes:

- [Section 4.6.1, "How to Upgrade Projects from Oracle CEP 2.1 to 10.3"](#)
- [Section 4.6.2, "How to Upgrade Projects from Oracle CEP 10.3 to 11g Release 1 \(11.1.1\)"](#)

For more information, see:

- [Section 3.4, "Configuring Eclipse"](#)
- *Oracle Complex Event Processing Getting Started*

4.6.1 How to Upgrade Projects from Oracle CEP 2.1 to 10.3

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

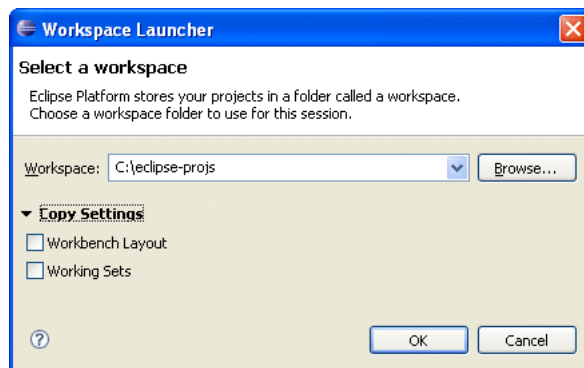
The following outlines the steps necessary to upgrade 2.1 projects to 10.3

To upgrade projects from Oracle CEP 2.1 to 10.3:

1. Open your Oracle CEP 2.1 project in Oracle CEP IDE for Eclipse.
2. Select **File > Switch Workspace > Other**.

The Workspace Launcher dialog appears as shown in [Figure 4–13](#).

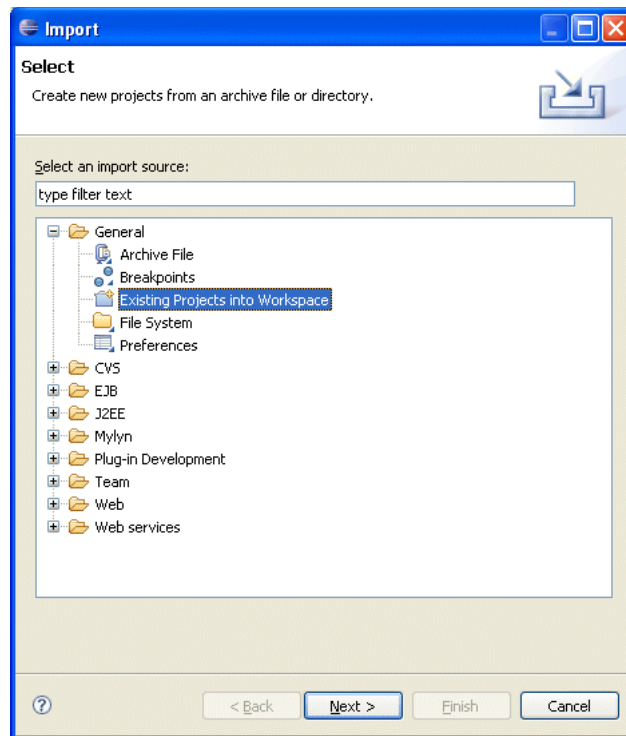
Figure 4–13 Workspace Launcher Dialog



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

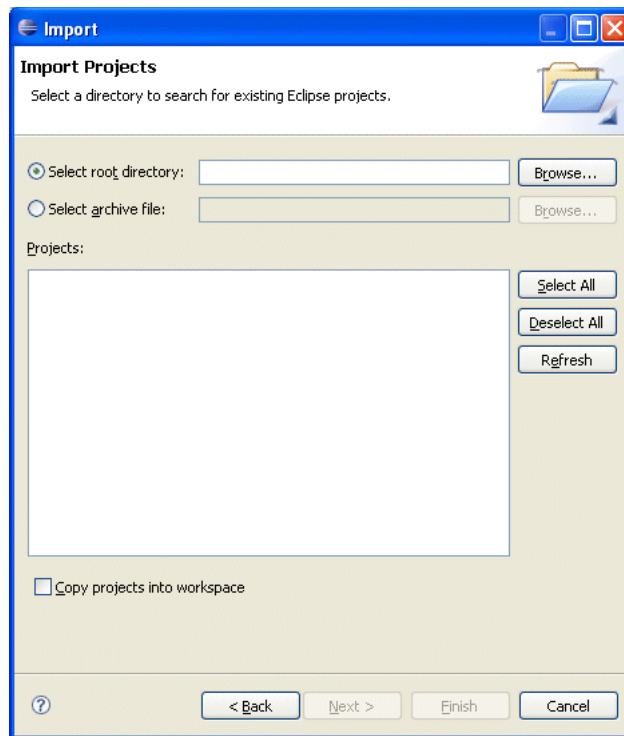
3. Click Browse and select a new workspace directory.
Eclipse exits and restarts using the new workspace.
4. Select **File > Import**.

The Import Dialog appears as shown in [Figure 4–14](#).

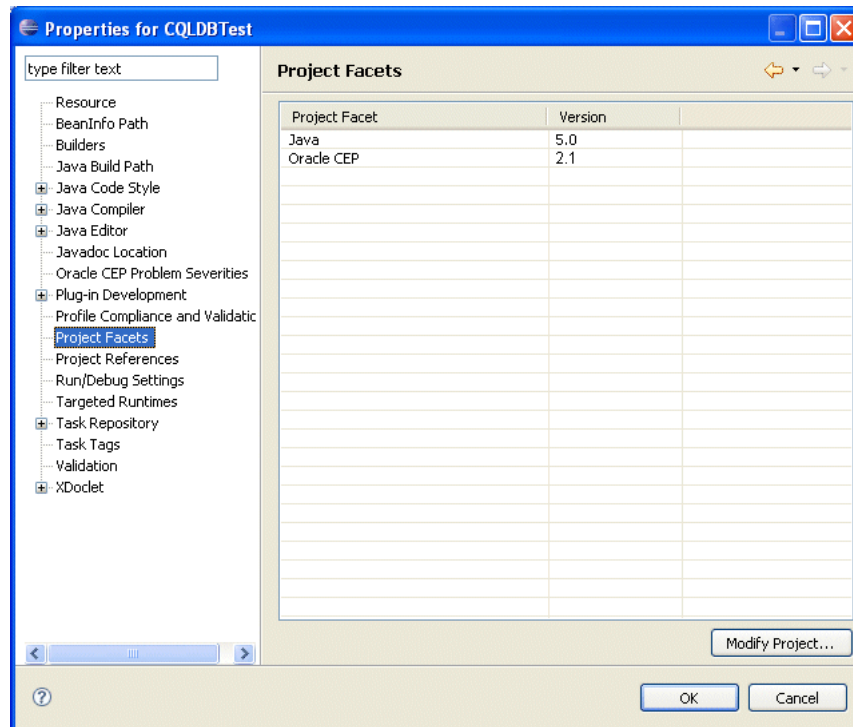
Figure 4–14 Import Dialog

5. Expand the **General** option and select **Existing Projects into Workspace**.
6. Click **Next**.

The Import Projects dialog appears as shown in [Figure 4–15](#).

Figure 4–15 Import Projects Dialog

7. Use the Import Projects dialog to import your 2.1 projects into the new workspace. Optionally, choose to copy the project files into your new workspace.
8. For each project, change the project facet version as follows:
 - a. Right-click your project and select **Properties**.
The Project Properties dialog appears as shown in [Figure 4–16](#).

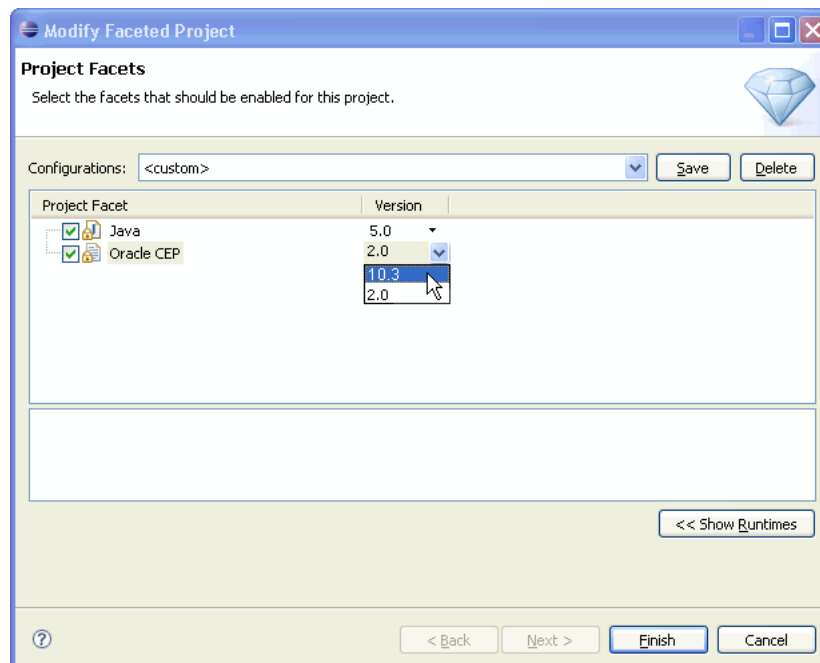
Figure 4–16 Project Properties Dialog: Project Facets

- b. Select the Project Facets option.

The Project Facet properties are displayed as [Figure 4–16](#) shows.

- c. Click **Modify Project**.

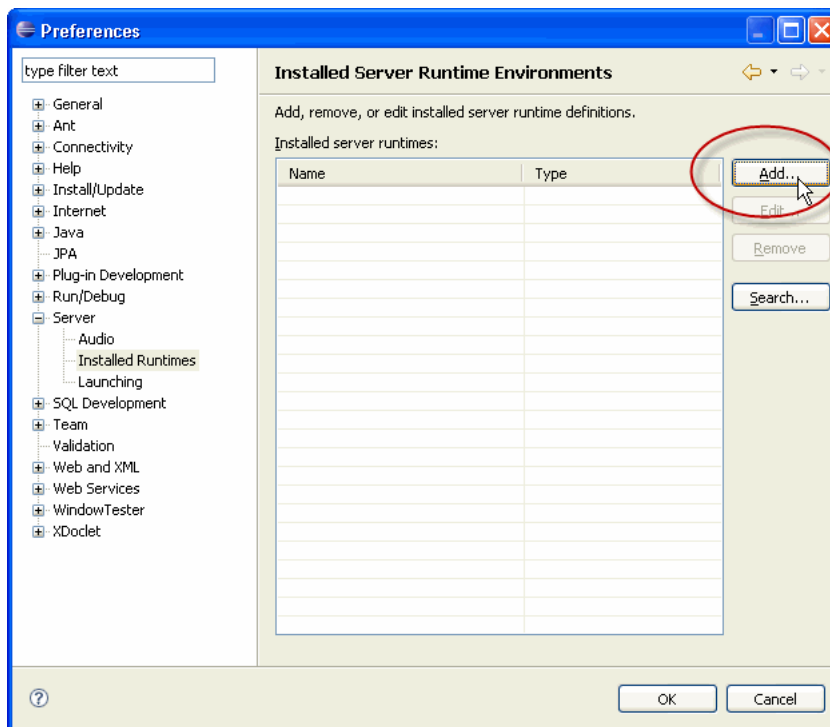
The Modify Faceted Project dialog appears shown in [Figure 4–17](#).

Figure 4–17 Modify Faceted Project

- d. For the Oracle CEP facet, select 10.3 from the **Version** pull-down menu.
 - e. Click **Finish**.
 - f. Click **OK**.
 - g. Repeat for the next project.
9. Create a new Oracle CEP server runtime:
- a. Select **Window > Preferences**.

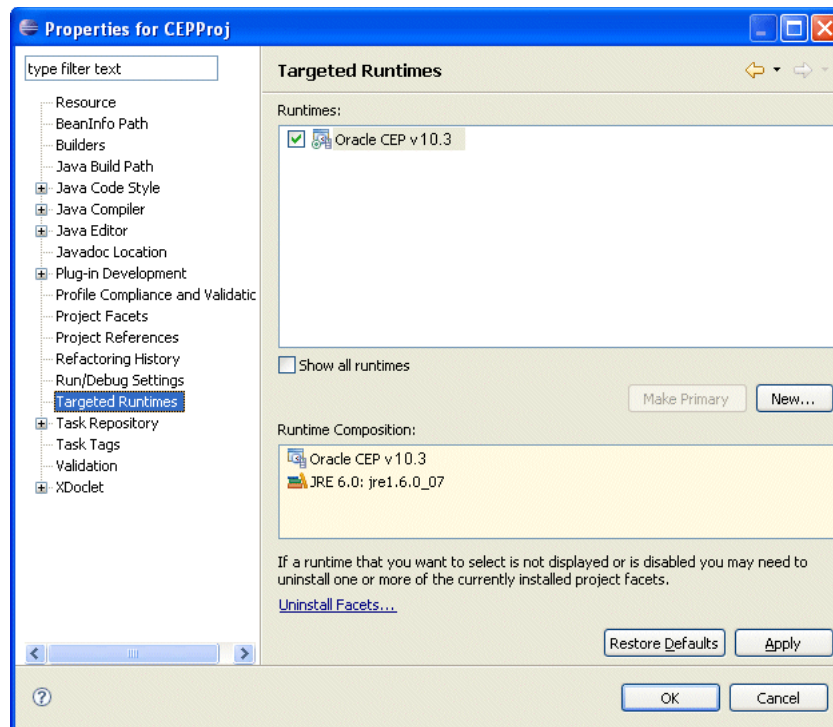
The Preferences dialog appears as shown in [Figure 4–18](#).

Figure 4–18 Preferences Dialog

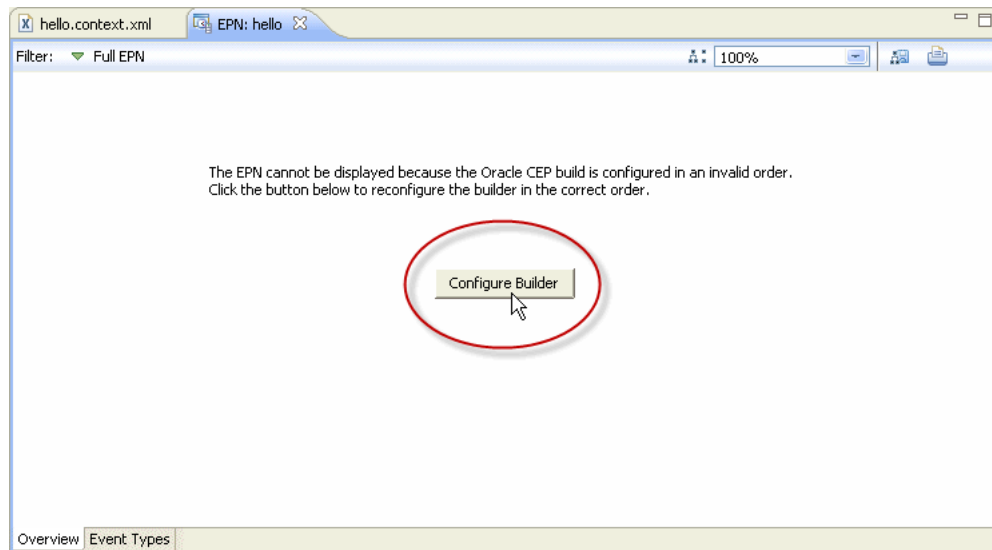


- b. Expand the **Server** option and select **Installed Runtimes**.
 - c. Add a new 10.3 Oracle CEP server runtime as [Section 5.2.3, "How to Create an Oracle CEP Server Runtime"](#) describes.
 - d. Click **OK**.
10. For each project, specify the new 10.3 Oracle CEP server runtime you created:
- a. Right-click your project and select **Properties**.

The Project Properties dialog appears as shown in [Figure 4–19](#).

Figure 4–19 Project Properties Dialog: Targeted Runtimes

- b. Select the **Targeted Runtimes** option.
The Targeted Runtimes properties are displayed as [Figure 4–19](#) shows.
 - c. Check the new Oracle CEP 10.3 targeted runtime you created.
 - d. Click **OK**.
 - e. Repeat for the next project.
11. For each project update the project builders:
 - a. Right-click the project and select **Open EPN Editor**.
 - b. If the EPN diagram opens without error, proceed to step 12.
 - c. If the EPN diagram opens with the error shown in [Figure 4–20](#), click the **Configure Builder** button.

Figure 4–20 Builder Error

The EPN diagram is displayed.

- d. Repeat for the next project.

12. Validate build inclusions:

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

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

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

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

13. Perform source changes, if necessary.

For more information, see:

- "Upgrading a WebLogic Event Server 2.0 Application to Run on Oracle CEP 10.3" in the *Oracle Complex Event Processing Getting Started*
- *Oracle CEP Release Notes* for 10.3 (http://download.oracle.com/docs/cd/E13157_01/wlevs/docs30/notes/notes.html)

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

4.6.2 How to Upgrade Projects from Oracle CEP 10.3 to 11g Release 1 (11.1.1)

While project structure has stayed the same since 10.3, the data stored in Oracle CEP Projects has changed significantly. It is therefore necessary to take steps to upgrade 10.3 projects manually before continuing their development in 11g Release 1 (11.1.1).

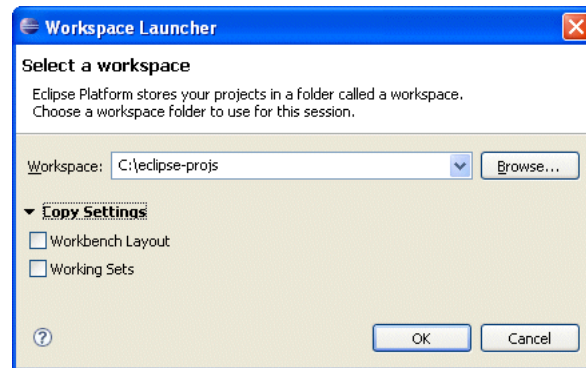
The following outlines the steps necessary to upgrade 10.3 projects to 11g Release 1 (11.1.1)

To upgrade projects from Oracle CEP 10.3 to 11g Release 1 (11.1.1)

1. Open your Oracle CEP 10.3 project in Oracle CEP IDE for Eclipse.
2. Select **File > Switch Workspace > Other**.

The Workspace Launcher dialog appears as shown in [Figure 4-21](#).

Figure 4-21 Workspace Launcher Dialog



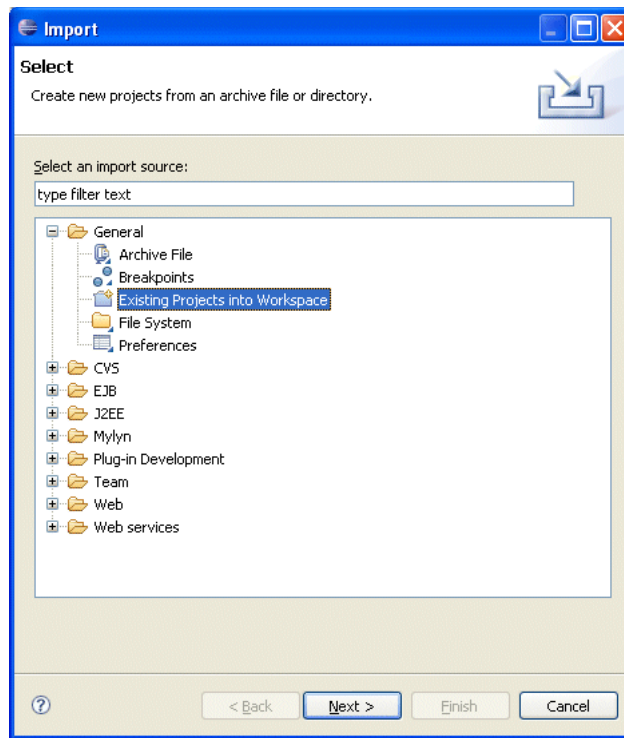
3. Click Browse and select a new workspace directory.

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

Eclipse exits and restarts using the new workspace.

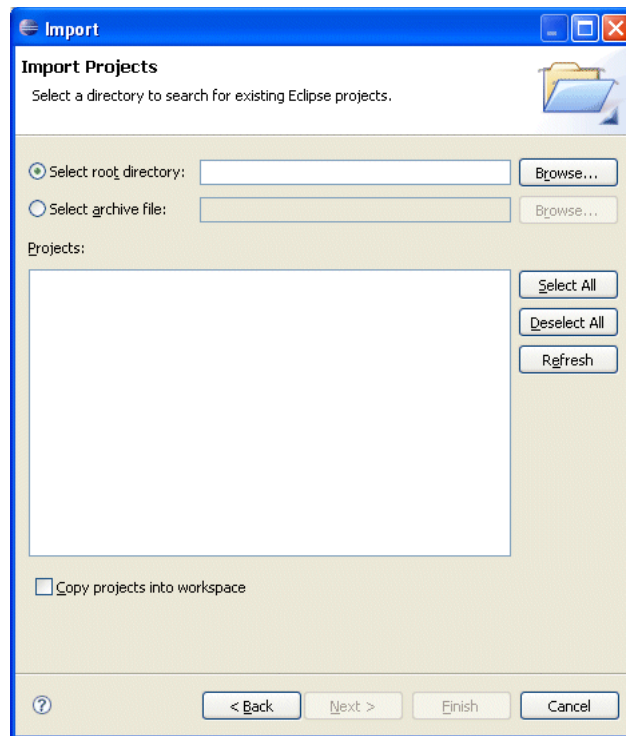
4. Select **File > Import**.

The Import Dialog appears as shown in [Figure 4-14](#).

Figure 4–22 Import Dialog

5. Expand the **General** option and select **Existing Projects into Workspace**.
6. Click **Next**.

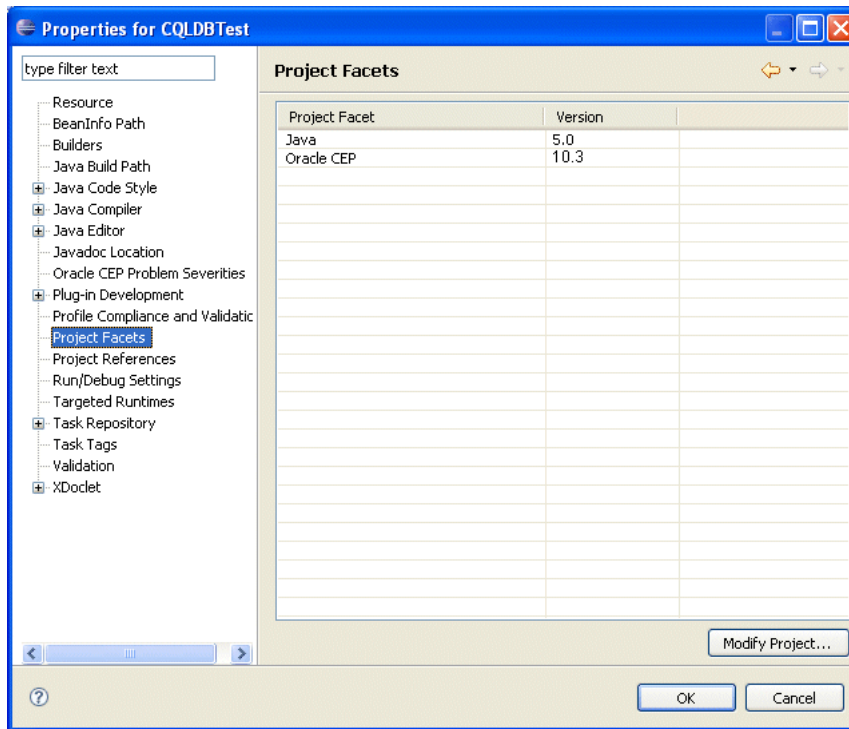
The Import Projects dialog appears as shown in [Figure 4–15](#).

Figure 4–23 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 4–16](#).

Figure 4–24 Project Properties Dialog: Project Facets



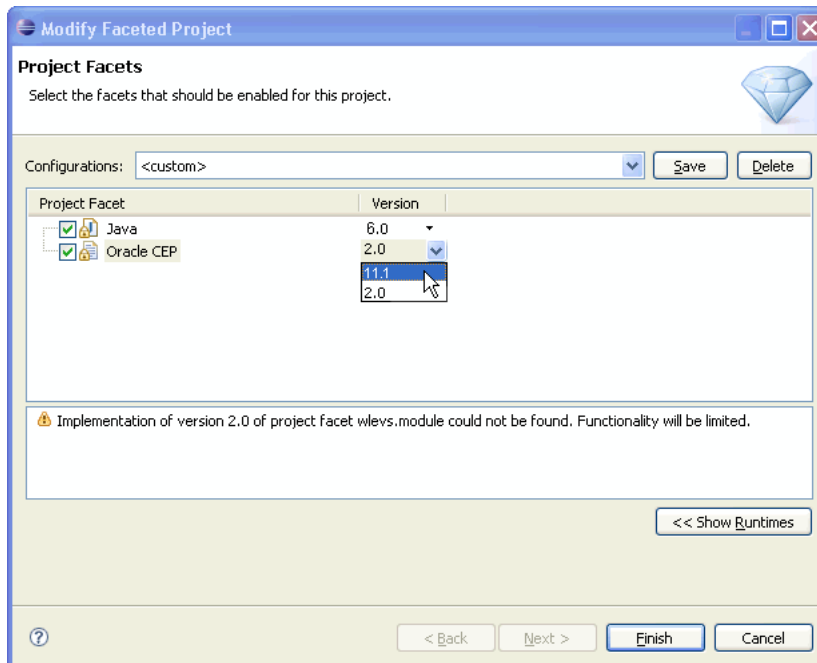
b. Select the Project Facets option.

The Project Facet properties are displayed as Figure 4–16 shows.

c. Click **Modify Project**.

The Modify Faceted Project dialog appears shown in Figure 4–17.

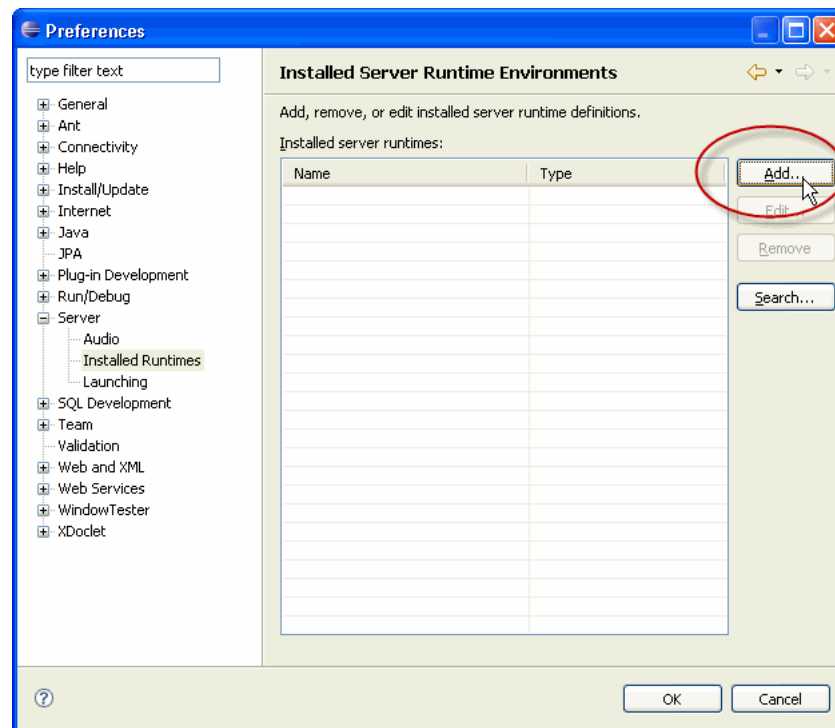
Figure 4–25 Modify Faceted Project



- d. For the Java facet, select 6.0 from the **Version** pull-down menu.
 - e. For the Oracle CEP facet, select 11.1 from the **Version** pull-down menu.
 - f. Click **Finish**.
 - g. Click **OK**.
 - h. Repeat for the next project.
9. Create a new Oracle CEP server runtime:
 - a. Select **Window > Preferences**.

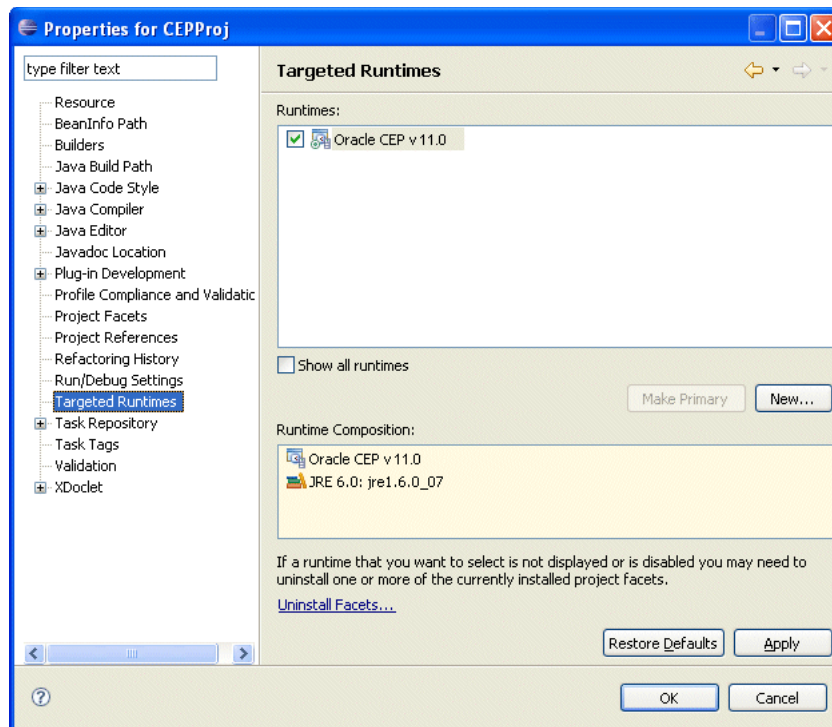
The Preferences dialog appears as shown in [Figure 4–18](#).

Figure 4–26 Preferences Dialog

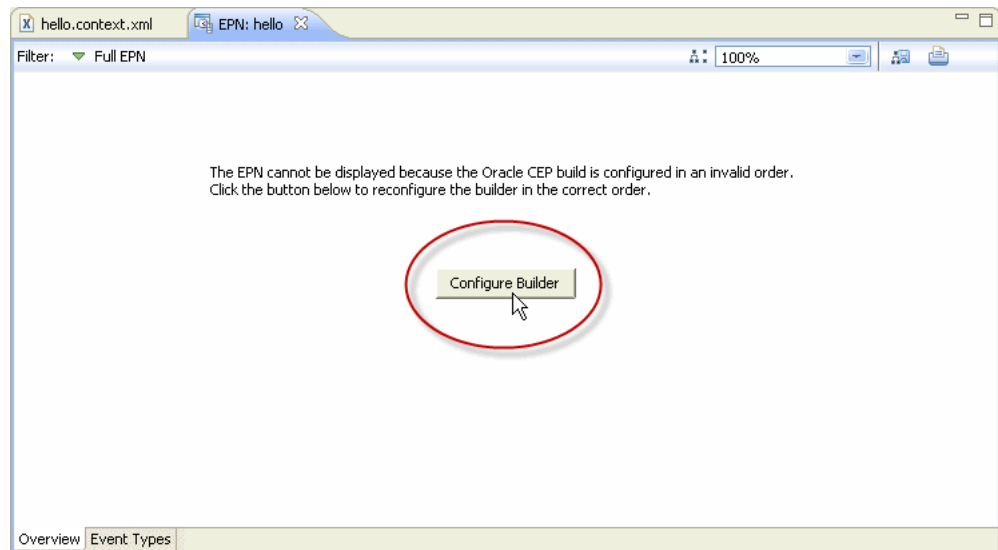


- b. Expand the **Server** option and select **Installed Runtimes**.
 - c. Add a new 11.0 Oracle CEP server runtime as [Section 5.2.3, "How to Create an Oracle CEP Server Runtime"](#) describes.
 - d. Click **OK**.
10. For each project, specify the new 11.0 Oracle CEP server runtime you created:
 - a. Right-click your project and select **Properties**.

The Project Properties dialog appears as shown in [Figure 4–19](#).

Figure 4–27 Project Properties Dialog: Targeted Runtimes

- b. Select the **Targeted Runtimes** option.
The Targeted Runtimes properties are displayed as [Figure 4–19](#) shows.
 - c. Check the new Oracle CEP 11.0 targeted runtime you created.
 - d. Click **OK**.
 - e. Repeat for the next project.
11. For each project update the project builders:
 - a. Right-click the project and select **Open EPN Editor**.
 - b. If the EPN diagram opens without error, proceed to step 12.
 - c. If the EPN diagram opens with the error shown in [Figure 4–20](#), click the **Configure Builder** button.

Figure 4–28 Builder Error

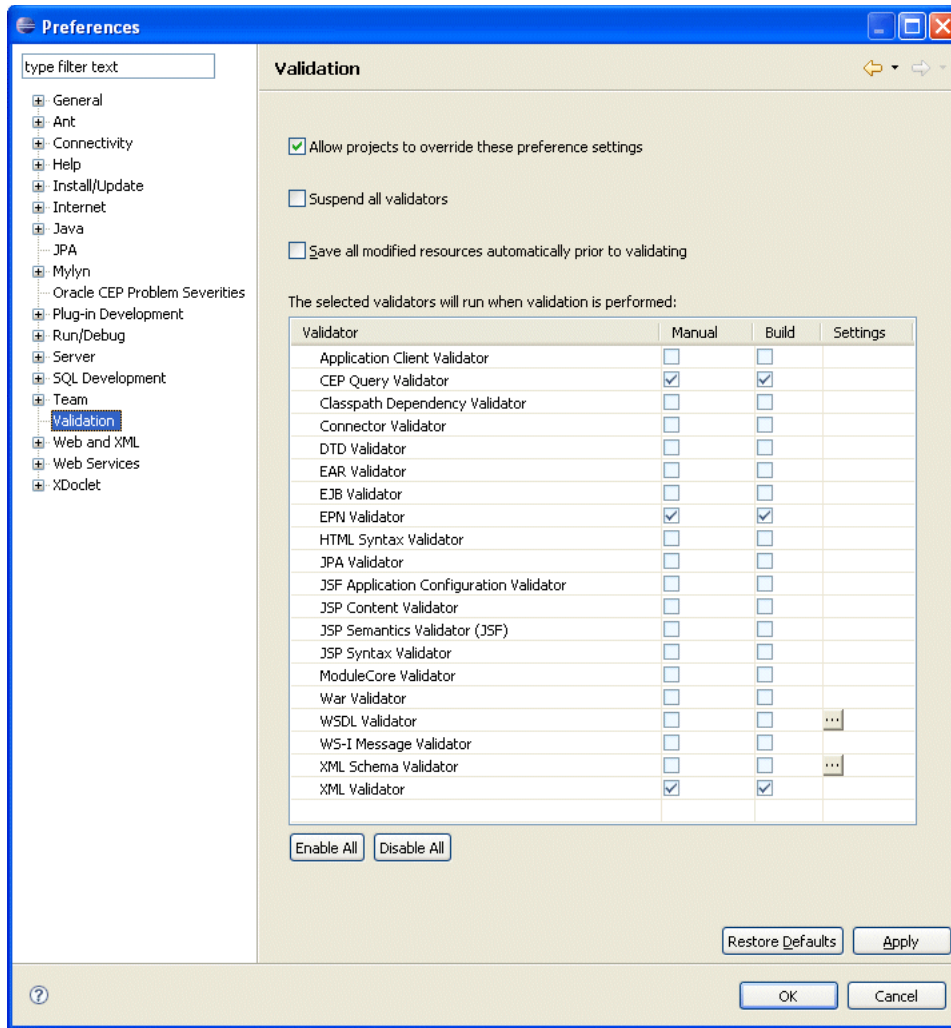
The EPN diagram is displayed.

d. Repeat for the next project.

12. Select **Window > Preferences**.

The Preferences dialog appears as shown in [Figure 4–29](#).

Figure 4–29 Preferences Dialog



13. Select the **Validaion** option.

14. Ensure that the following validation options are checked:

- CQL Validator
- EPN Validator
- XML Validator

15. Unselect all other options.

16. Click **OK**.

17. Validate build inclusions:

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

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

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

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

```
bundle.properties, \
```

```
.
```

18. Perform source changes, if necessary.

For more information, see:

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

4.7 Managing Libraries and Other Non-Class Files in Oracle CEP 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 CEP 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 CEP IDE for Eclipse and you can check the library into a source code control system as part of the project.

For more information, see:

- [Section 4.7.1, "How to Add a Standard JAR File to an Oracle CEP Project"](#)
- [Section 4.7.2, "How to Add an OSGi Bundle to an Oracle CEP Project"](#)
- [Section 4.7.3, "How to Add a Property File to an Oracle CEP Project"](#)
- [Section 4.7.4, "How to Export a Package"](#)
- [Section 4.7.5, "How to Import a Package"](#)
- [Section 24.2.2.2.2, "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 CEP 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 4.7.2, "How to Add an OSGi Bundle to an Oracle CEP Project"](#)
- [Section 24.1.3, "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 CEP project so that the property file is deployed with your application and is available at runtime.

For more information, see:

- [Section 4.7.3, "How to Add a Property File to an Oracle CEP Project"](#)
- [Section 1.1.5.1, "Accessing Component and Server Configuration Using the ConfigurationPropertyPlaceholderConfigurer Class"](#)

4.7.1 How to Add a Standard JAR File to an Oracle CEP Project

If the library you need to use is a standard JAR file, you can add it to your Oracle CEP project. Alternatively, you can add a library as an OSGi bundle (see [Section 4.7.2, "How to Add an OSGi Bundle to an Oracle CEP Project"](#)).

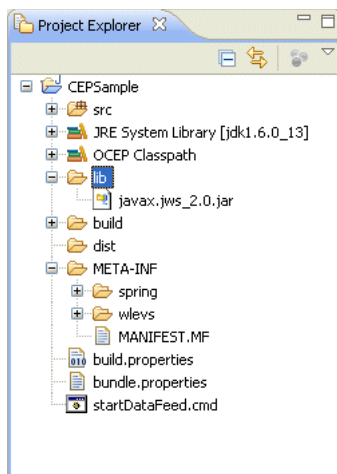
When you add a standard JAR file to an Oracle CEP project, you can optionally expose some or all of its packages to other bundles that will depend on this bundle.

To add a standard JAR file to an Oracle CEP project:

1. Create a folder in your Oracle CEP 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 CEP IDE for Eclipse, copy your JAR file into the `lib` folder.
3. Inside the Oracle CEP IDE for Eclipse, right-click the `lib` folder and select **Refresh**.

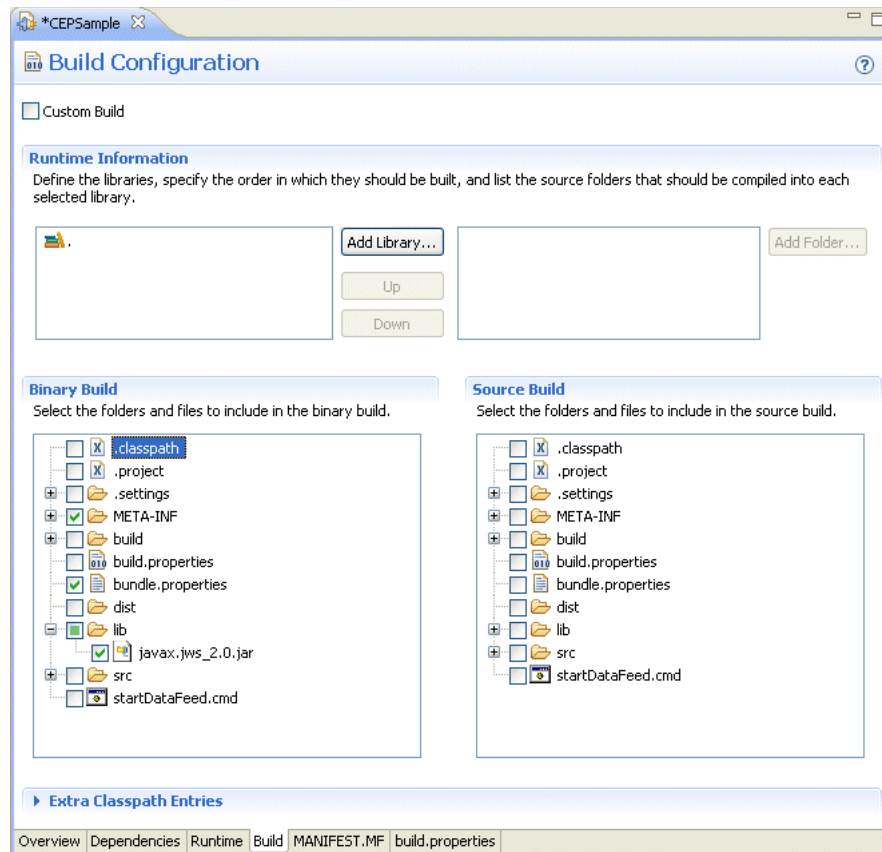
The JAR file appears in the `lib` folder as [Figure 4–30](#) shows.

Figure 4–30 Oracle CEP 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 4–31](#) shows.

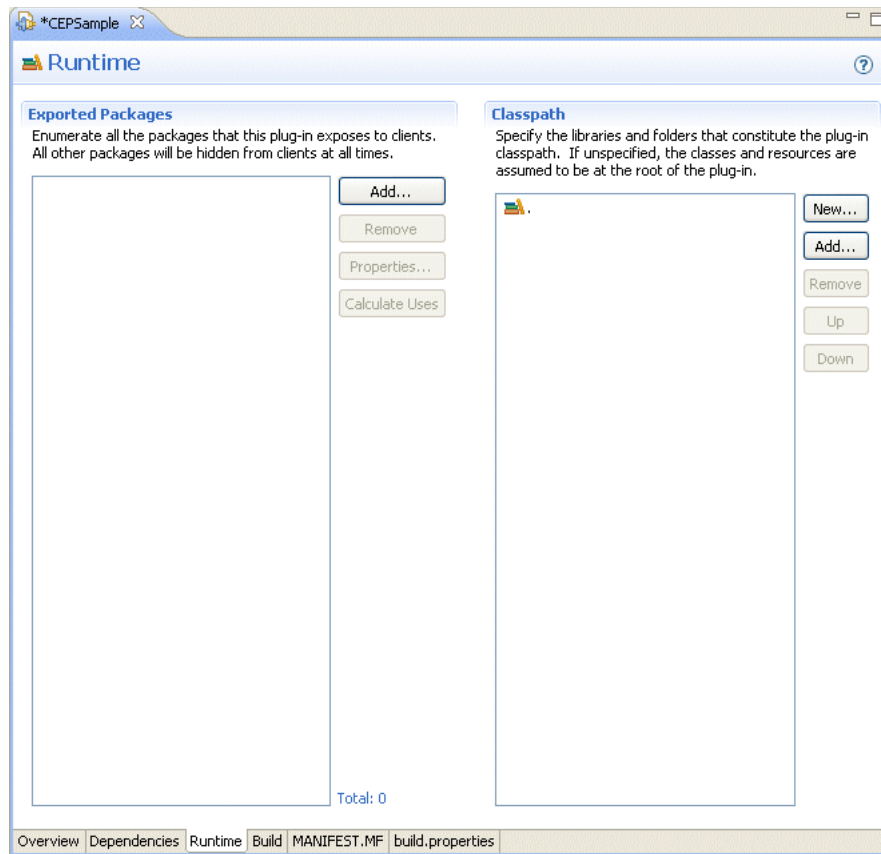
Figure 4–31 Manifest Editor: Build Tab

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 4–31](#) shows.
 - Press hit `CTRL-SHIFT-S` to save all files.

This edits the `build.properties` file in your project, and tells the Oracle CEP 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 4–32](#) shows.

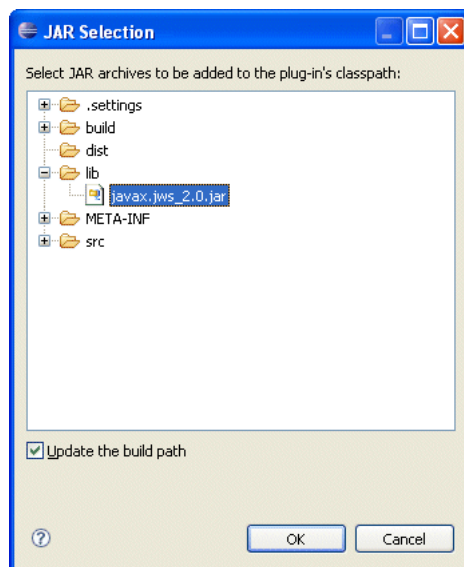
Figure 4–32 Manifest Editor - Runtime Tab



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 4–33](#).

Figure 4–33 JAR Selection Dialog



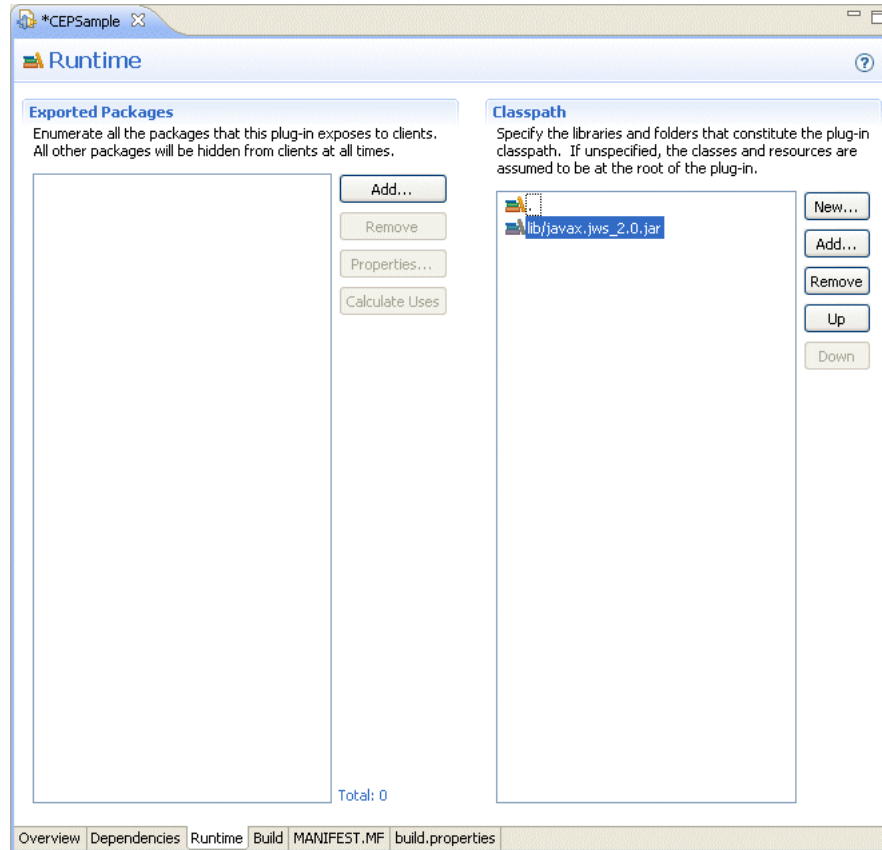
- Select the JAR you want to add to the bundle.

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 4–34](#) shows.

Figure 4–34 Manifest Editor Runtime tab After Adding a JAR to the Classpath

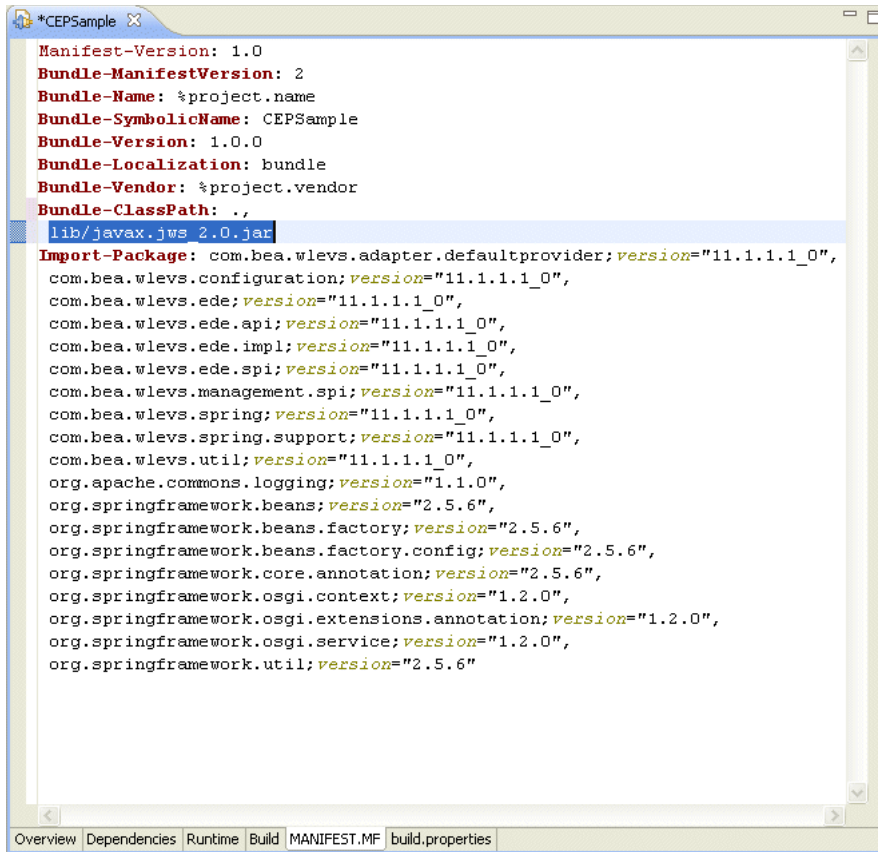


- Press hit **CTRL-SHIFT-S** to save all files.

This edits the `MANIFEST.MF` file, putting the JAR on your project classpath.

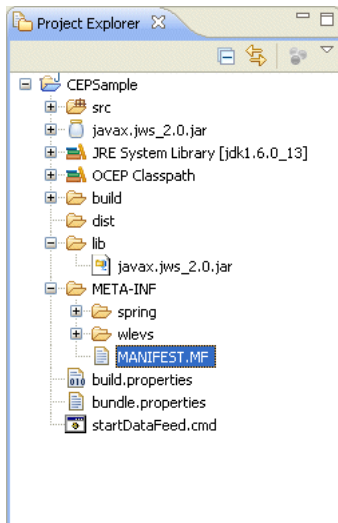
To confirm this, in the Manifest Editor, click the **MANIFEST.MF** tab and note that the JAR is now listed under the `Bundle-Classpath` property as [Figure 4–35](#) shows.

Figure 4–35 Manifest Editor MANIFEST.MF Tab



Note also that the JAR now appears as a library at the root of the project as [Figure 4–36](#) shows.

Figure 4–36 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 4.7.4, "How to Export a Package"](#) describes.

4.7.2 How to Add an OSGi Bundle to an Oracle CEP Project

If the library you need to use is an OSGi bundle, you can add it to your Oracle CEP project. Alternatively, you can add a library as a standard JAR file (see [Section 4.7.1, "How to Add a Standard JAR File to an Oracle CEP Project"](#)).

To add an OSGi bundle to an Oracle CEP 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 CEP server manually. For more information, see [Section 24.1.3, "Application Libraries"](#).

To add an OSGi bundle to an Oracle CEP project:

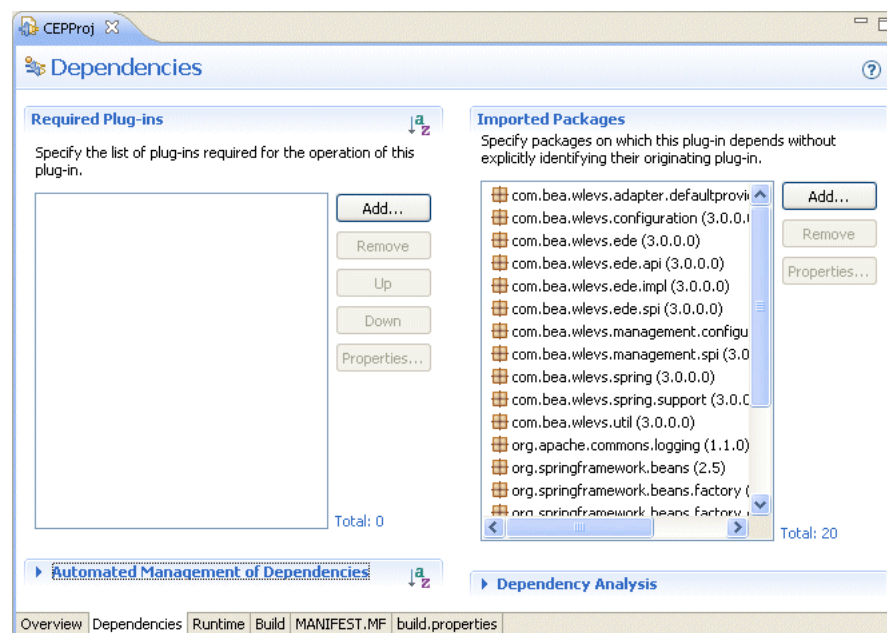
1. Place the OSGi bundle in the `DOMAIN_DIR/servername/modules` directory, where `DOMAIN_DIR` refers to the domain directory such as `/oracle_cep/user_projects/domains/mydomain` and `servername` refers to the server instance, such as `myserver`. For example:

```
c:\oracle_cep\user_projects\domains\mydomain\myserver\modules
```

2. Start the Oracle CEP 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 4–37](#) shows.

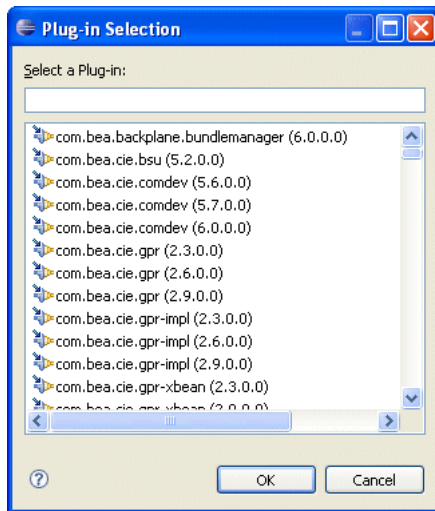
Figure 4–37 Manifest Editor: Dependencies Tab



5. Click the **Dependencies** tab.
6. In the **Required Plug-ins** area, click **Add**.

The Plug-in Selection dialog appears as shown in [Figure 4–38](#).

Figure 4–38 Plug-in Selection Dialog



7. Select the bundle you added to the `DOMAIN_DIR/servername/modules` directory of your Oracle CEP server installation directory in step 1 and click **OK**.

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

4.7.3 How to Add a Property File to an Oracle CEP Project

You can add a Java property file to an Oracle CEP project so that the property file is deployed with your application and is available at runtime.

To add a property file to an Oracle CEP project:

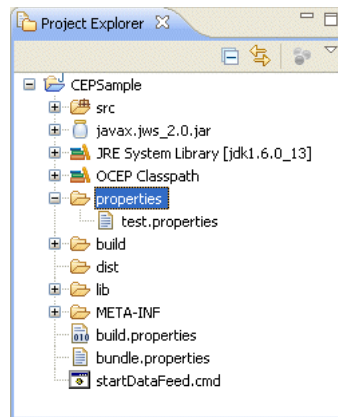
1. Create a folder in your Oracle CEP IDE for Eclipse project to put the property files in.

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**.

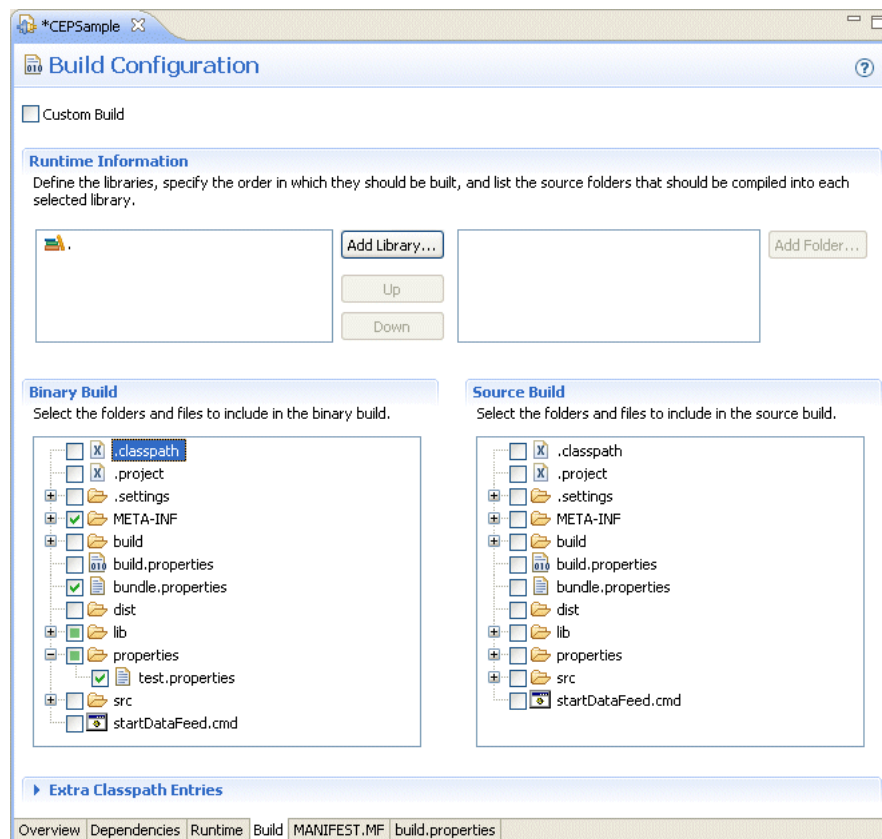
2. Outside of the Oracle CEP IDE for Eclipse, copy your property file into the `properties` folder.
3. Inside the Oracle CEP IDE for Eclipse, right-click the `properties` folder and select **Refresh**.

The property file appears in the `properties` folder as [Figure 4–39](#) shows.

Figure 4–39 Oracle CEP IDE for Eclipse properties 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 4–40](#) shows.

Figure 4–40 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 4–40](#) shows.

- Press hit CTRL-SHIFT-S to save all files.

This edits the `build.properties` file in your project, and tells the Oracle CEP 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 4-1](#) shows:

Example 4-1 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"));
    }
}
```

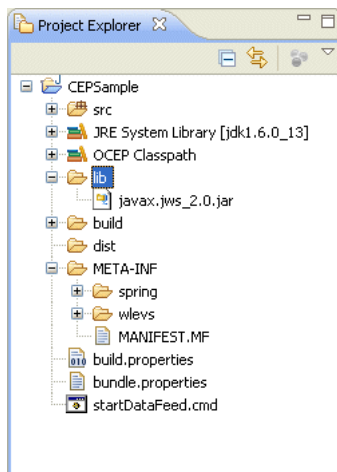
4.7.4 How to Export a Package

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:

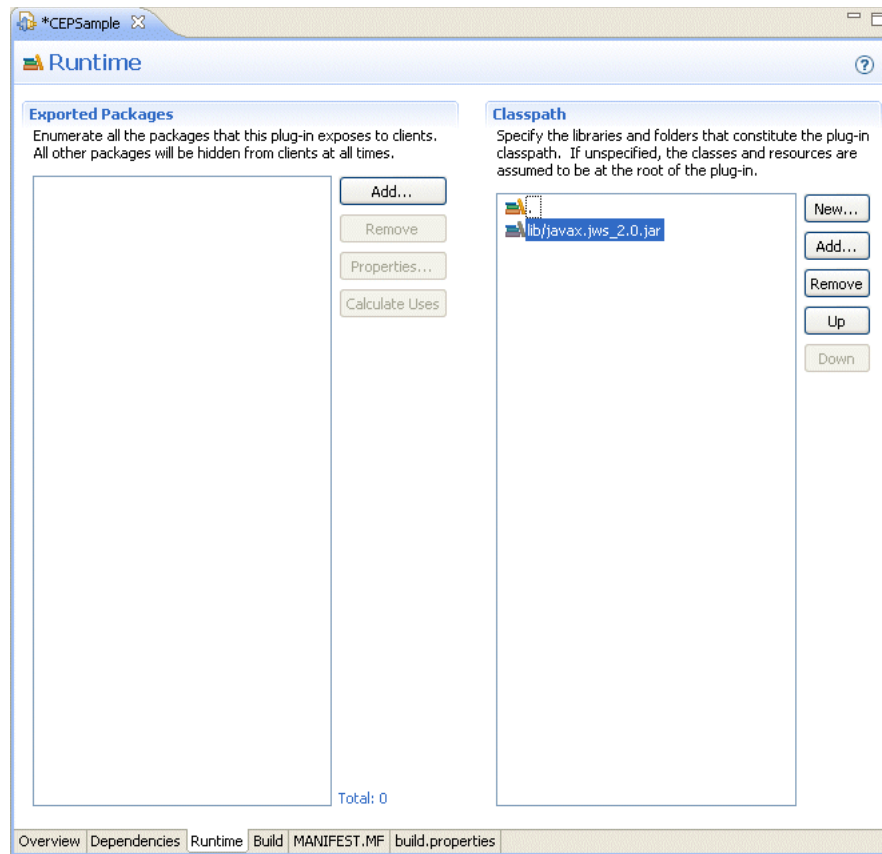
1. Inside the Oracle CEP IDE for Eclipse, expand the `META-INF` directory as [Figure 4-41](#) shows.

Figure 4-41 Oracle CEP IDE for Eclipse lib Directory

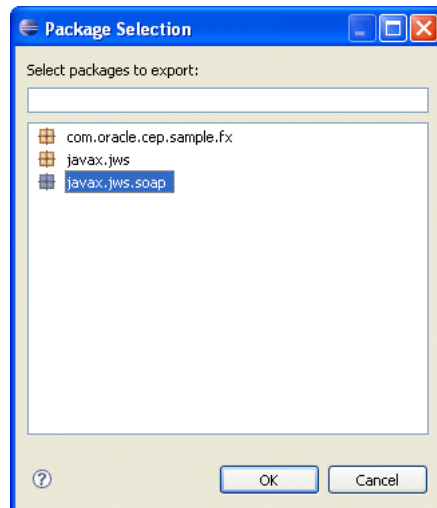


2. Right-click the `MANIFEST.MF` file and select **Open With > Plug-in Manifest Editor**.

The Manifest Editor opens as [Figure 4-42](#) shows.

Figure 4–42 Manifest Editor: Runtime tab


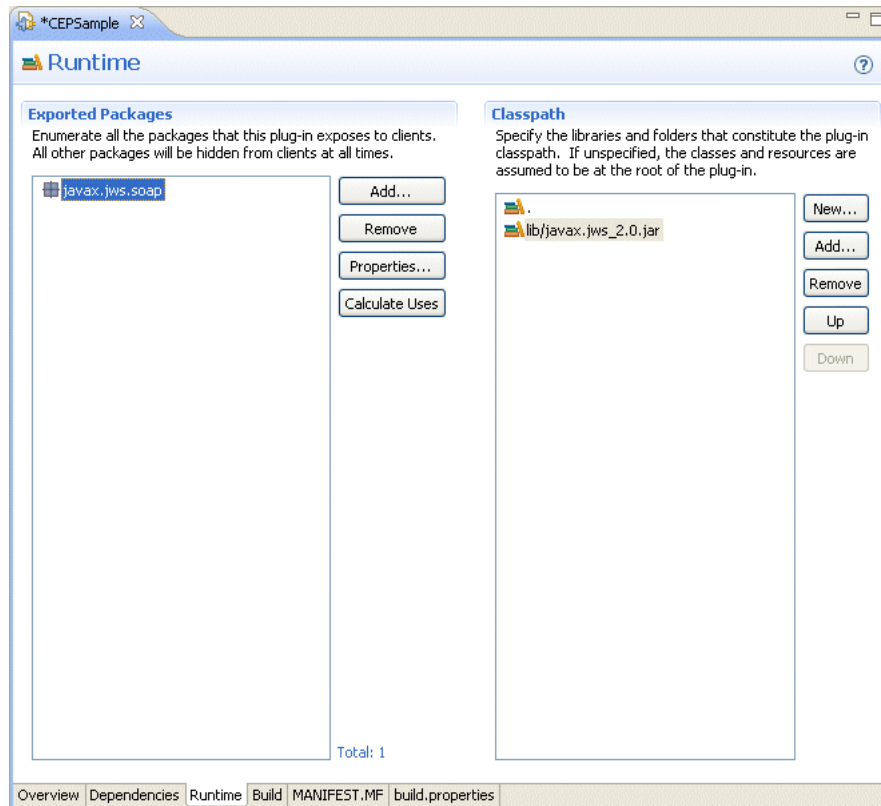
3. Click the **Runtime** tab.
The Runtime tab appears as [Figure 4–42](#) shows.
4. In the Exported Packages area, click the **Add** button.
The Package Selection dialog appears as [Figure 4–43](#) shows.

Figure 4–43 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 4–44](#) shows.

Figure 4–44 Manifest Editor Runtime tab After Exporting a Package



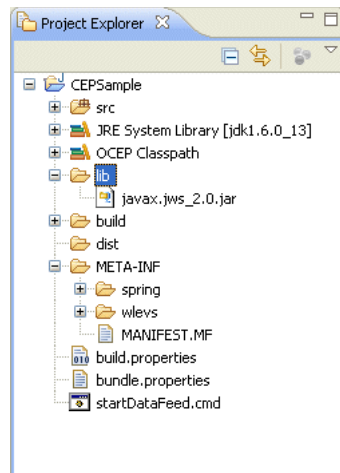
- 7. Press **CTRL-SHIFT-S** to save all files.

4.7.5 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.

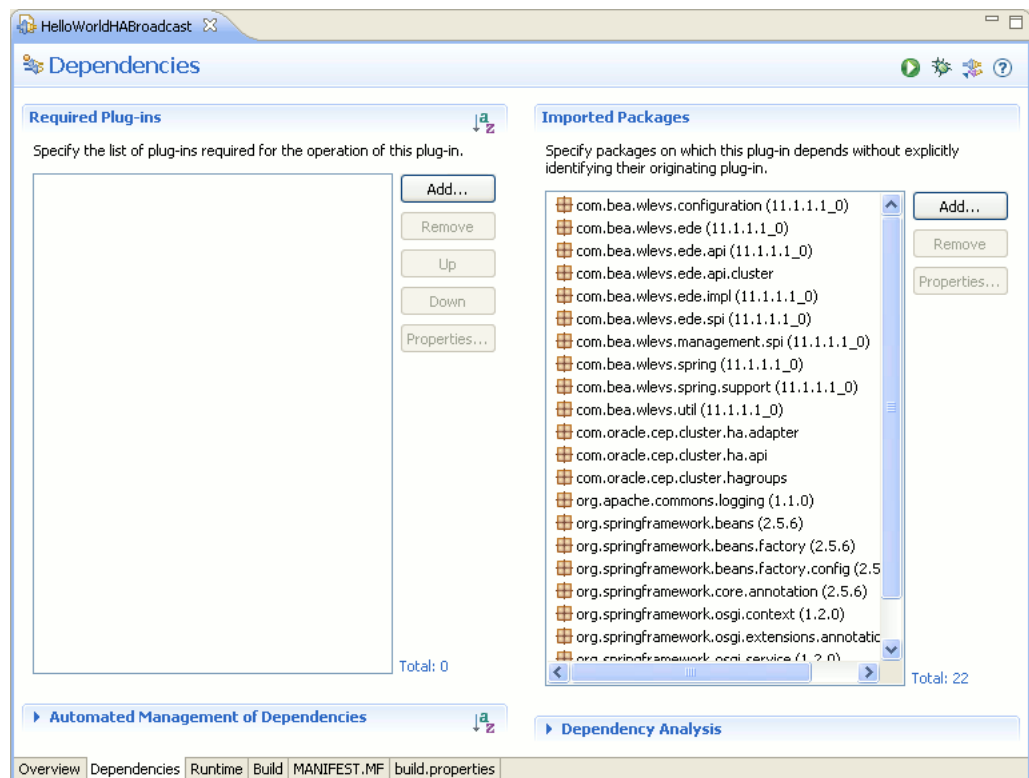
To import a package:

- 1. Inside the Oracle CEP IDE for Eclipse, expand the `META-INF` directory as [Figure 4–45](#) shows.

Figure 4–45 Oracle CEP IDE for Eclipse lib Directory


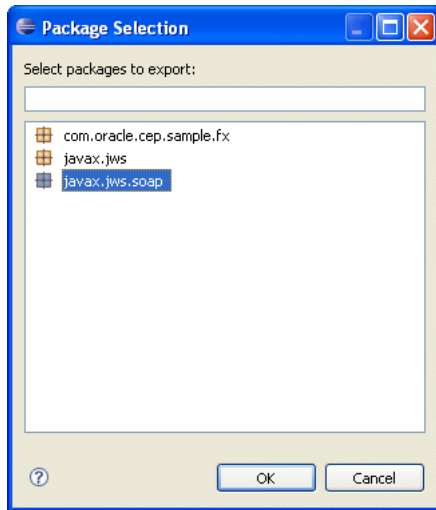
2. Right-click the `MANIFEST.MF` file and select **Open With > Plug-in Manifest Editor**.

The Manifest Editor opens as [Figure 4–46](#) shows.

Figure 4–46 Manifest Editor: Dependencies tab


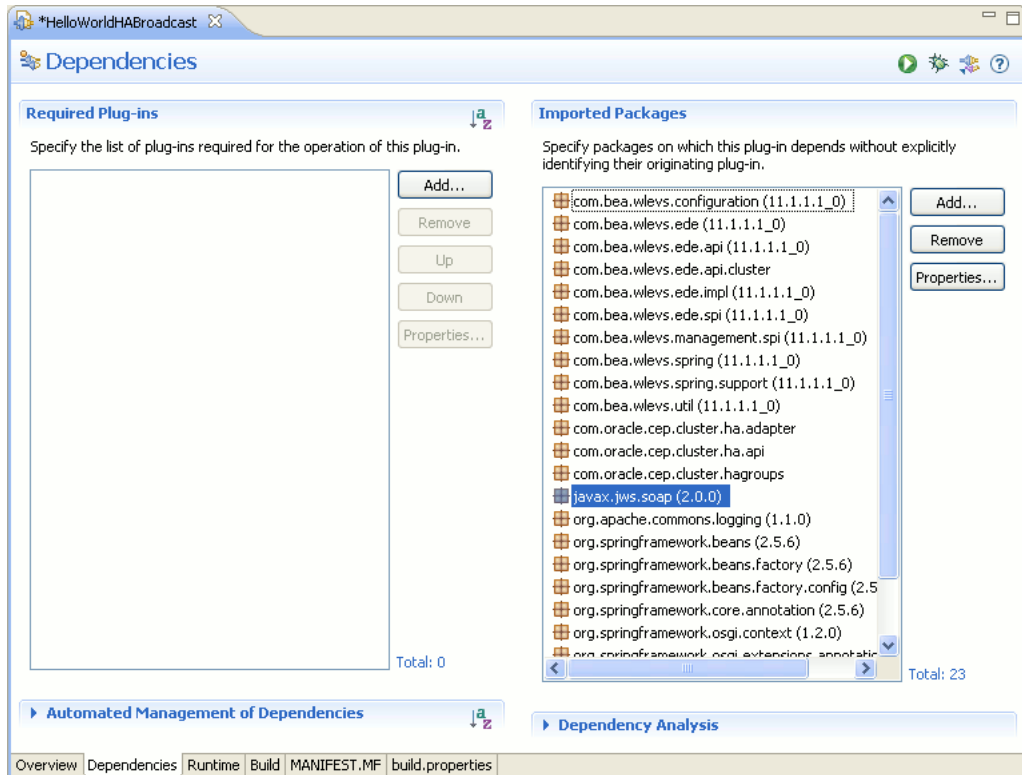
3. Click the **Dependencies** tab.
The Dependencies tab appears as [Figure 4–46](#) shows.
4. In the Imported Packages area, click the **Add** button.
The Package Selection dialog appears as [Figure 4–47](#) shows.

Figure 4–47 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 4–48](#) shows.

Figure 4–48 Manifest Editor Dependencies tab After Importing a Package



7. Press **CTRL-SHIFT-S** to save all files.

4.8 Configuring Oracle CEP IDE for Eclipse Preferences

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

- [Section 4.8.1, "How to Configure Application Library Path Preferences"](#)
- [Section 4.8.2, "How to Configure Problem Severity Preferences"](#)

4.8.1 How to Configure Application Library Path Preferences

You can define the path to an Oracle CEP server domain directory that contains application libraries that extend the Oracle CEP runtime.

For more information, see [Section 24.3.1, "How to Define the Application Library Directory Using Oracle CEP IDE for Eclipse"](#).

4.8.2 How to Configure Problem Severity Preferences

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

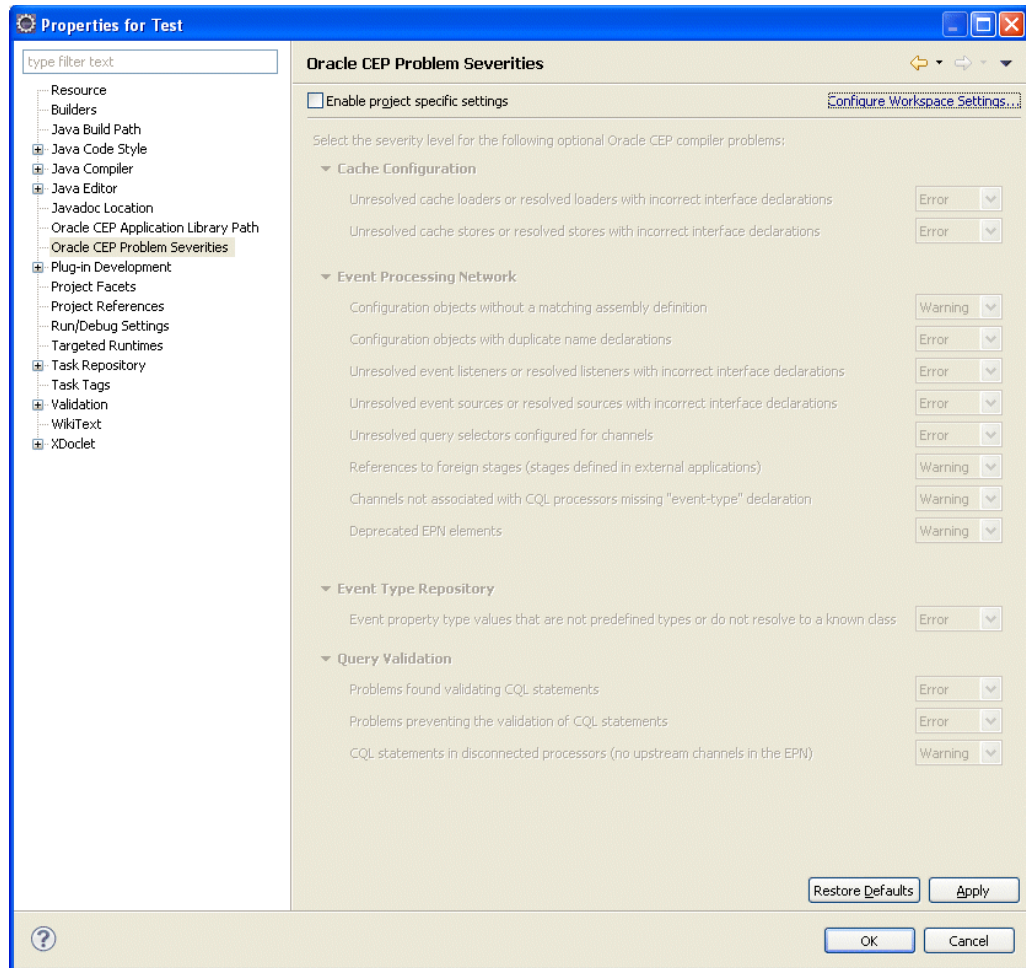
You can configure these preferences for each project individually or you can configure these preferences the same for all the projects in a given workspace.

To configure problem severity preferences:

1. Open the EPN Editor (see [Section 6.1, "Opening the EPN Editor"](#))
2. Select **Window > Preferences**.

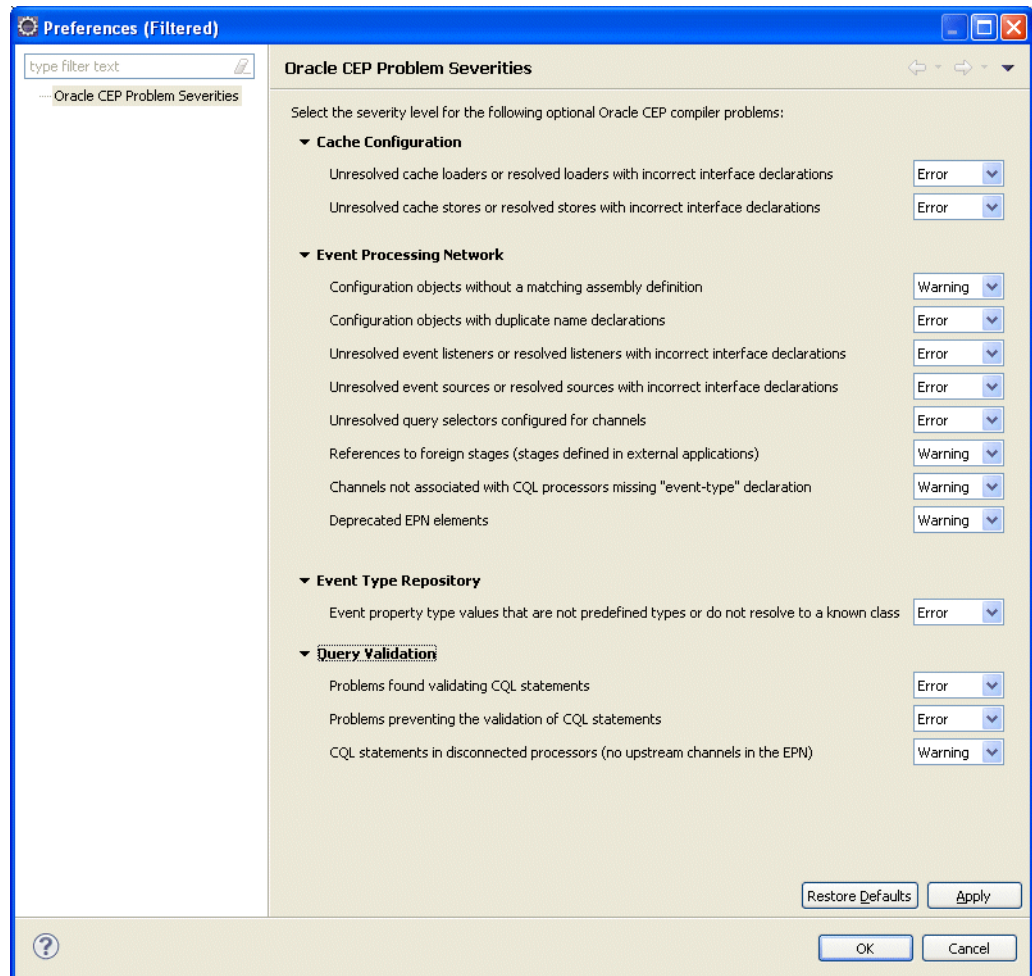
The Preferences dialog appears as shown in [Figure 4-49](#).

Figure 4–49 Properties Dialog



3. Select **Oracle CEP Problem Severities**.
4. Choose the configuration scope:
 - To configure problem severities for all the projects in a workspace, click **Configure Workspace Settings**.

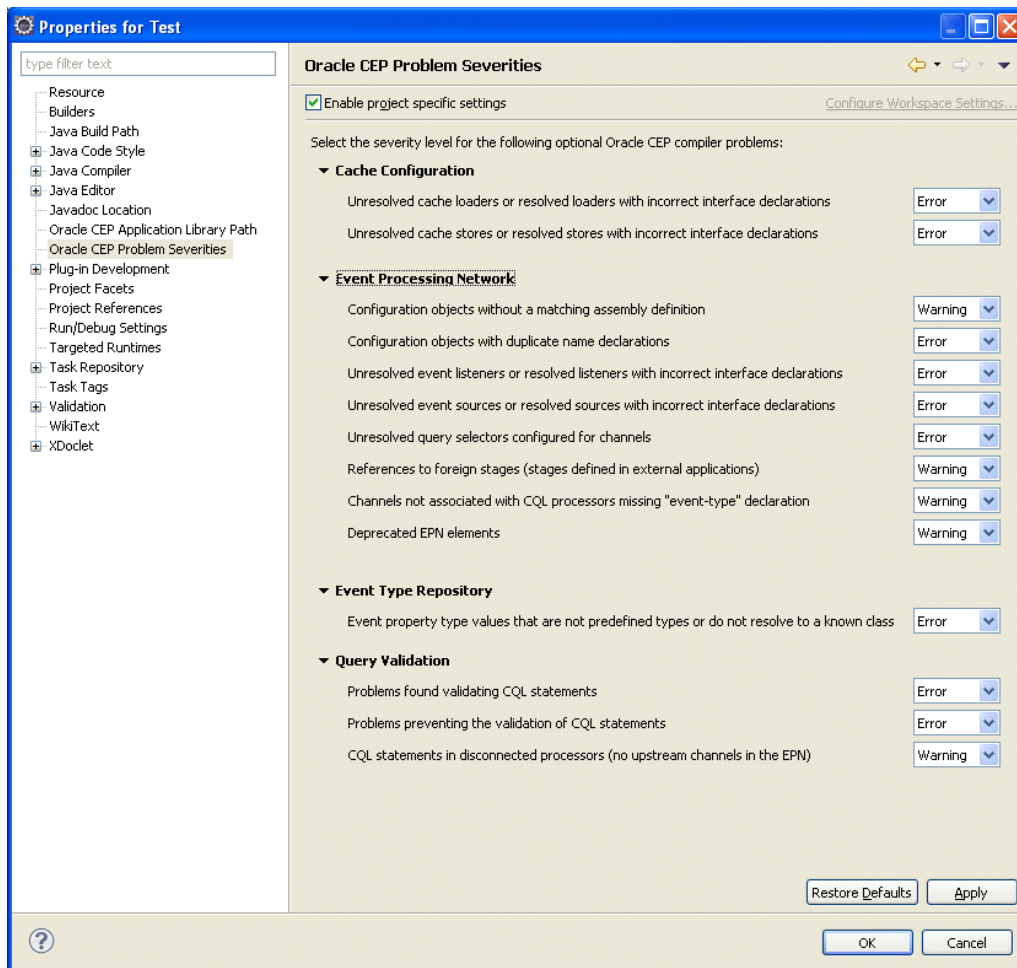
The Oracle CEP Problem Severities dialog appears as [Figure 4–50](#) shows.

Figure 4–50 Oracle CEP Problem Severities Dialog: Workspace

- To configure problem severities for the current project only, click **Enable project specific settings**.

The Oracle CEP Problem Severities dialog appears as shown in [Figure 4–51](#).

Figure 4–51 Oracle CEP Problem Severities Dialog: Project



5. Select a severity for each type of problem. You can select one of:
 - Error: treat the problem as an error.
 - Warning: treat the problem as a warning.
 - Ignore: ignore the problem.

Table 4–7 describes each of the problem areas you can configure.

Table 4-7 Oracle CEP Problem Severities

Category	Problem	Description
Cache Configuration	Unresolved cache loaders or resolved loaders with incorrect interface declarations.	Ensure that the assembly file contains a <code>bean</code> 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"> Section 12.2.3, "Configuring an Oracle CEP Local Cache Loader" Section 12.3.4.1, "Configuring an Oracle Coherence Cache Loader."
	Unresolved cache stores or resolved stores with incorrect interface declarations.	Ensure that the assembly file contains a <code>bean</code> 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"> Section 12.2.4, "Configuring an Oracle CEP Local Cache Store" Section 12.3.4.2, "Configuring an Oracle Coherence Cache Store."
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 CEP configuration files are identified by a name. Validate that no two configuration elements in an application have the same name.
	Unresolved event listeners or resolved listeners with incorrect interface declarations	An event processing network is built by defining how elements in the network pull or push to events to other elements in the application. Validate that the target of an event push, a listener declaration on an EPN assembly element, implements the interfaces required to receive pushed events.
	Unresolved event sources or resolved sources with incorrect interface declarations	An event processing network is built by defining how elements in the network pull or push to events to other elements in the application. Validate that the source of an event pull, a source declaration on an EPN assembly element, implements the interfaces required to provide pulled events.
	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 D.78, "selector" .
	References to foreign stages (stages defined in external applications).	Ensure that references to foreign stages can be resolved. For more information, see Section 1.1.1.9, "Foreign Stages" .
	Channels not associated with 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 9, "Configuring Channels" .
	Deprecated EPN elements	Oracle CEP 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 Section 1.1.2, "Oracle CEP 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 4–7 (Cont.) Oracle CEP 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.

6. Click **Apply**.
7. Click **OK**.

Oracle CEP IDE for Eclipse and Oracle CEP Servers

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

This chapter describes:

- [Section 5.1, "Oracle CEP Server Overview"](#)
- [Section 5.2, "Creating Oracle CEP Servers"](#)
- [Section 5.3, "Managing Oracle CEP Servers"](#)
- [Section 5.4, "Debugging an Oracle CEP Application Running on an Oracle CEP Server"](#)

5.1 Oracle CEP Server Overview

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

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

[Table 5–1](#) maps Eclipse terminology used by the Oracle CEP IDE for Eclipse to Oracle CEP server terminology.

Table 5–1 *Eclipse and Oracle CEP Server Concepts*

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

Table 5–1 (Cont.) Eclipse and Oracle CEP Server Concepts

Eclipse IDE Concept	Oracle CEP Server Concept	Description
Server and Server Instance	Domain	<p>The Oracle CEP IDE for Eclipse uses the term Server to describe an actual runnable Oracle CEP server instance. You can think of it as something that has start scripts, for example. In Oracle CEP server terminology, this equates to a Domain. When you set up a server, you specify the domain that this instance will run.</p> <p>For more information on domains, see:</p> <ul style="list-style-type: none"> ■ For more information, see "Administrating Oracle CEP Standalone-Server Domains" in the <i>Oracle Complex Event Processing Administrator's Guide</i>. ■ For more information, see "Administrating Oracle CEP Standalone-Server Domains" in the <i>Oracle Complex Event Processing Administrator's Guide</i>.
Publish	Deploy	The Oracle CEP IDE for Eclipse typically uses the term Publish to describe physically deploying an application to a server.
Project	Application or Deployment	A project in the Oracle CEP IDE for Eclipse becomes a single Oracle CEP application packaged as an OSGi bundle. It is deployed to a server and shows in the Oracle CEP server's <code>deployments.xml</code> file.

Server definitions are the central concept in controlling an Oracle CEP server from the Oracle CEP IDE for Eclipse. It is from the server definition that you start and stop the server. After associating a project with the server, you can publish (deploy) the application to and unpublish (undeploy) the application from the server, all without having to leave the Oracle CEP IDE for Eclipse. For more information, see [Section 5.2, "Creating Oracle CEP Servers"](#).

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

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

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

For more information, see:

 - [Section 5.3.1, "How to Start a Local Oracle CEP Server"](#)
 - [Section 5.3.2, "How to Stop a Local Oracle CEP Server"](#)
- Attach to a running Oracle CEP server.

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

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

For more information, see:

- [Section 5.3.3, "How to Attach to an Existing Local Oracle CEP Server Instance"](#)
- [Section 5.3.5, "How to Detach From an Existing Oracle CEP Server Instance"](#)

5.2 Creating Oracle CEP Servers

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

You can create a local or remote Oracle CEP server:

- **Local server:** a local Oracle CEP server is one in which both the server and server runtime are on the same host
- **Remote server:** a remote Oracle CEP 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 CEP IDE for Eclipse).

This section describes:

- [Section 5.2.1, "How to Create a Local Oracle CEP Server and Server Runtime"](#)
- [Section 5.2.2, "How to Create a Remote Oracle CEP Server and Server Runtime"](#)
- [Section 5.2.3, "How to Create an Oracle CEP Server Runtime"](#)

5.2.1 How to Create a Local Oracle CEP 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 CEP 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.

For more information, see:

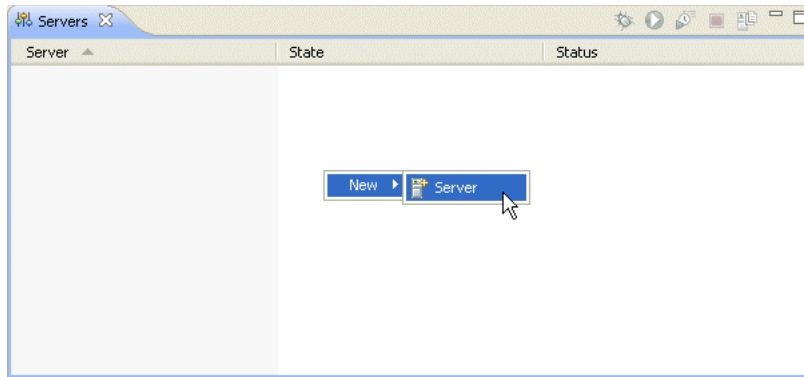
- [Section 5.2.3, "How to Create an Oracle CEP Server Runtime"](#)
- [Section 5.2.2, "How to Create a Remote Oracle CEP Server and Server Runtime"](#)

To create a local Oracle CEP server and server runtime:

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

The Servers view appears as shown in [Figure 5–1](#).

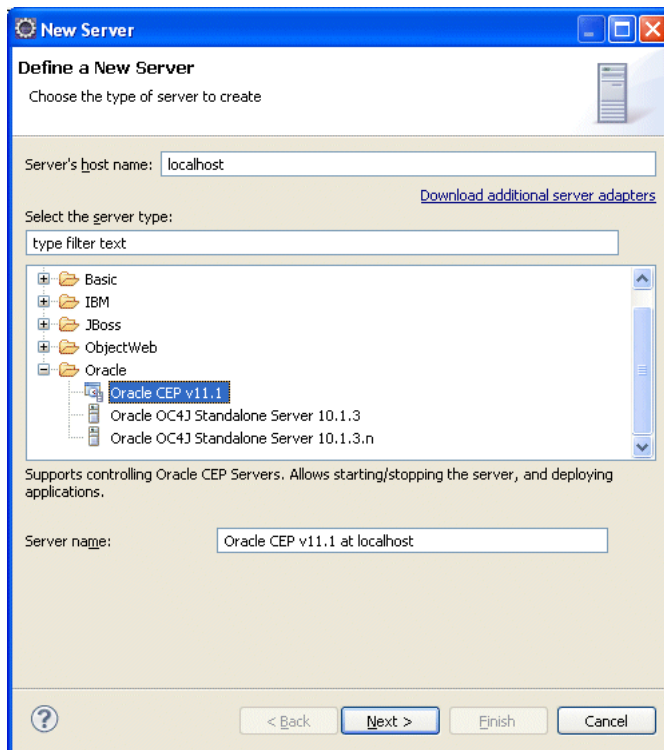
Figure 5–1 Oracle CEP IDE for Eclipse Server View



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 CEP 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 CEP 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 CEP server, there will be no installed server runtimes:

In this case, the New Server: Define New Server dialog appears as [Figure 5–2](#) shows.

Figure 5–2 New Server: Define New Server Dialog (No Installed Runtimes)



Configure the new server as follows:

- a. Configure the dialog as shown in [Table 5-2](#).

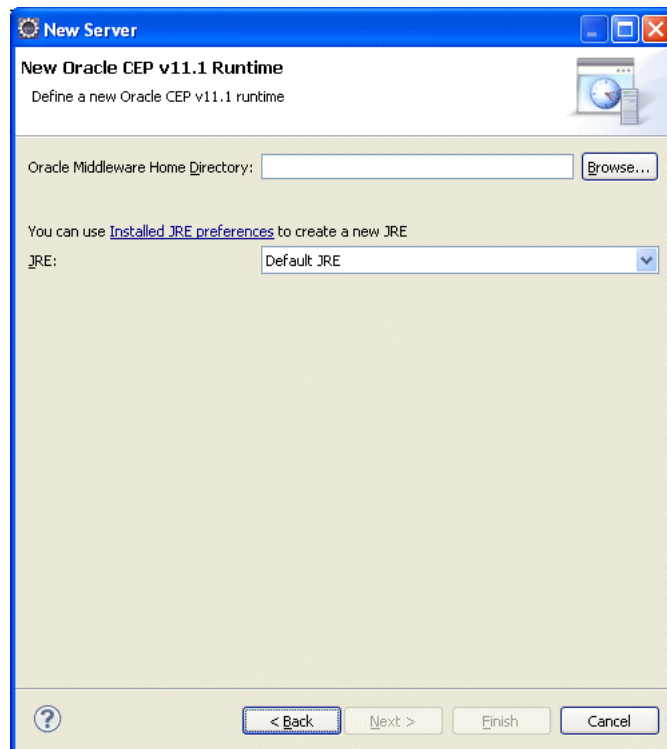
Table 5-2 New Server: Define New Server Dialog (No Installed Runtimes) Attributes

Attribute	Description
Server's host name	The host name of the computer on which you installed Oracle CEP server. For development, this will typically be <code>localhost</code> .
Select the server type	The type of Oracle CEP server. In this example, choose <code>Oracle CEP v11</code>
Server name	The name of this Oracle CEP server. Default: <code>Oracle CEP v11.1 at HOSTNAME</code> Where <code>HOSTNAME</code> is the value you entered in the Server's host name field.

- b. Click **Next**.

The New Server: New Oracle CEP v11 Runtime dialog appears as shown in [Figure 5-3](#).

Figure 5-3 New Server: New Oracle CEP v11.1 Runtime Dialog



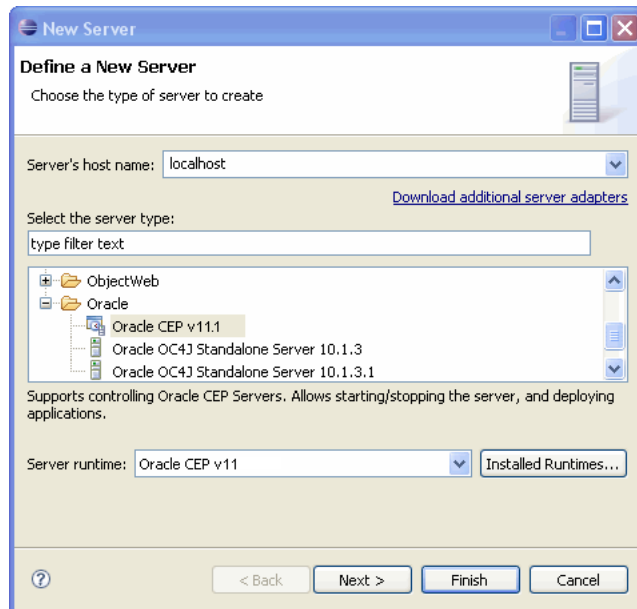
- c. Configure the dialog as shown in [Table 5-3](#).

Table 5–3 New Server: New Oracle CEP v11 Runtime Dialog Attributes

Attribute	Description
Oracle Middleware Home Directory	<p>The fully qualified path to the Oracle CEP server installation directory. This is the same as the "Middleware Home" that was selected when installing the server.</p> <p>When selecting this directory, select the directory that contains the Oracle CEP installation rather than the Oracle CEP 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 CEP installation directory.</p> <p>For more information, see "Oracle Fusion Middleware Directory Structure and Concepts" in the <i>Oracle Complex Event Processing Getting Started</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 Installed JRE preferences link to create a new JRE.</p> <p>Be sure to choose a Java 6 JRE.</p> <p>NOTE: The Oracle CEP server JRE is ultimately set by the JAVA_HOME 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 CEP server, there will be one or more installed server runtimes.
- In this case, the New Server: Define New Server dialog appears as [Figure 5–4](#) shows.

Figure 5–4 New Server: Define New Server (Installed Runtimes) Dialog



Configure the dialog as shown in [Table 5-4](#).

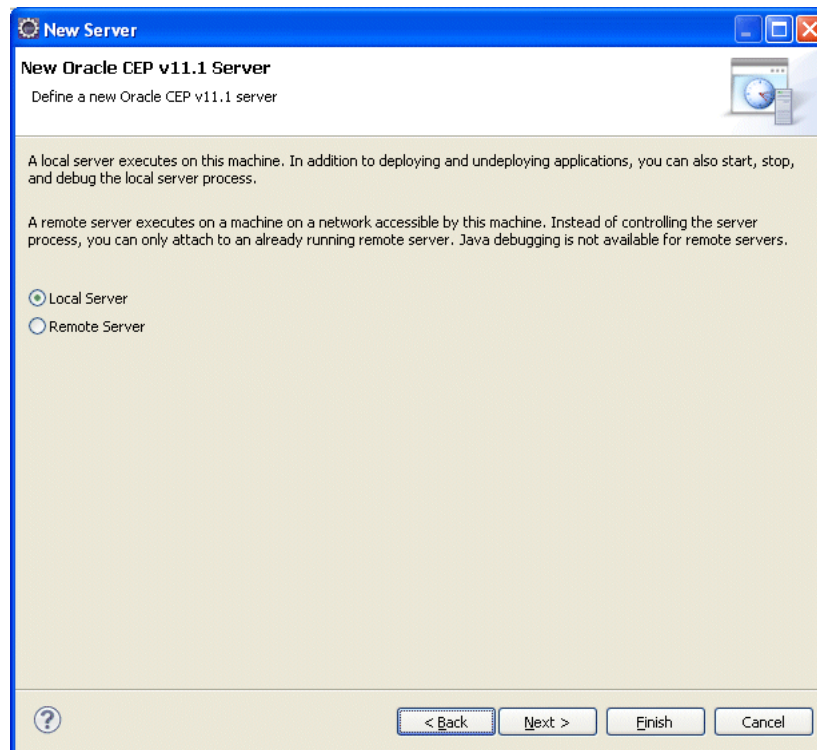
Table 5-4 New Server: Define New Server (Installed Runtimes) Dialog Attributes

Attribute	Description
Server's host name	The host name of the computer on which you installed Oracle CEP server. For development, this will typically be localhost.
Select the server type	The type of Oracle CEP server. In this example, choose Oracle CEP v11.
Server runtime	Select the server runtime from the pull-down menu. To create or edit server runtimes, click Installed Runtimes . For more information, see Section 5.2.3, "How to Create an Oracle CEP Server Runtime" .

6. Click Next.

The New Server: New Oracle CEP v11.1 Server dialog appears as [Figure 5-5](#) shows.

Figure 5-5 New Server: New Oracle CEP v11.1 Server

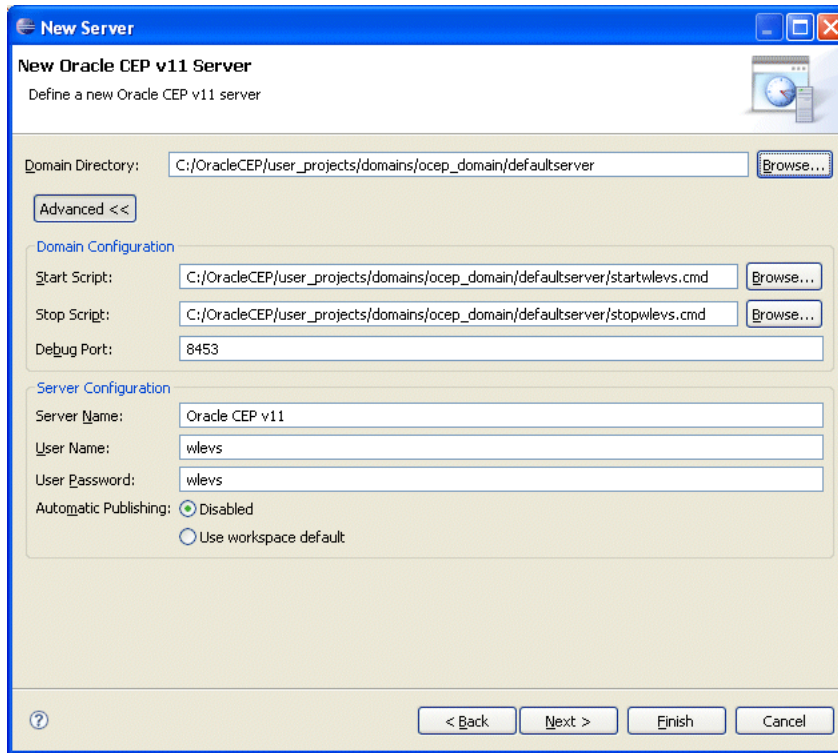


7. Select Local Server.

8. Click Next.

The New Server: New Oracle CEP v11 Server dialog appears as [Figure 5-6](#) shows.

Figure 5–6 New Server: New Oracle CEP v11 Server Dialog for a Local Server



9. Click **Advanced** and configure the dialog as shown in [Table 5–5](#).

Table 5–5 New Server: New Oracle CEP v11 Server Dialog Attributes for a Local Server

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

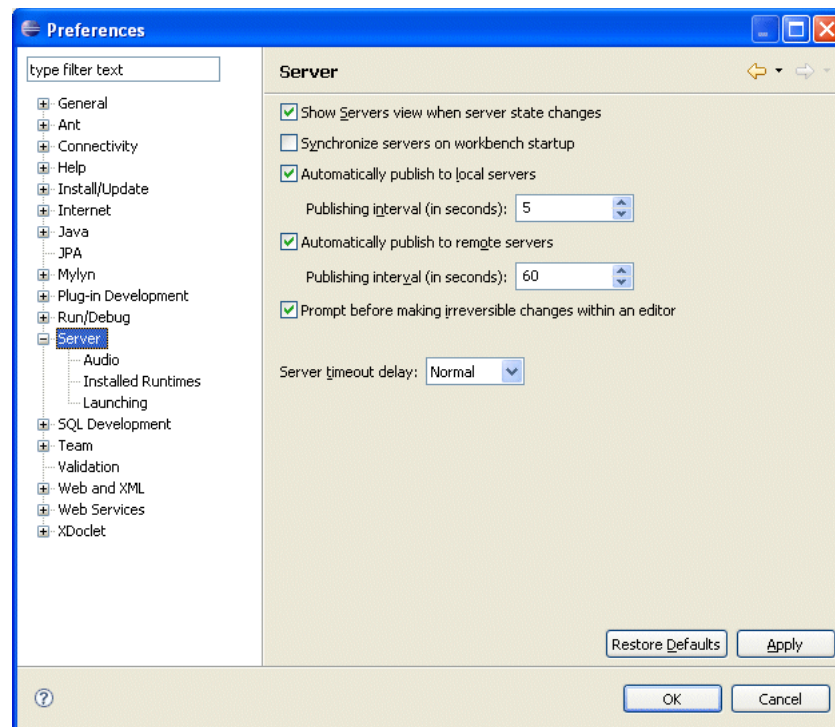
Table 5–5 (Cont.) New Server: New Oracle CEP v11 Server Dialog Attributes for a Local

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

10. Click **Finish**.

11. If you configured **Automatic Publishing to Use Workspace Default**, select **Windows > Preferences**.

The Preferences dialog appears as [Figure 5–7](#) shows.

Figure 5–7 Preferences - Server

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 CEP IDE for Eclipse publishes changes to the server (in seconds).
Default: 60 seconds.
- **Automatically publish to remote servers:** enable or disable this option, as required.
Default: enabled.
 - **Publishing interval:** configure the frequency at which the Oracle CEP IDE for Eclipse publishes changes to the server (in seconds).
Default: 60 seconds.

14. Click OK.

5.2.2 How to Create a Remote Oracle CEP 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 CEP 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 CEP IDE for Eclipse).

Alternatively, you can create a local server and server runtime.

For more information, see:

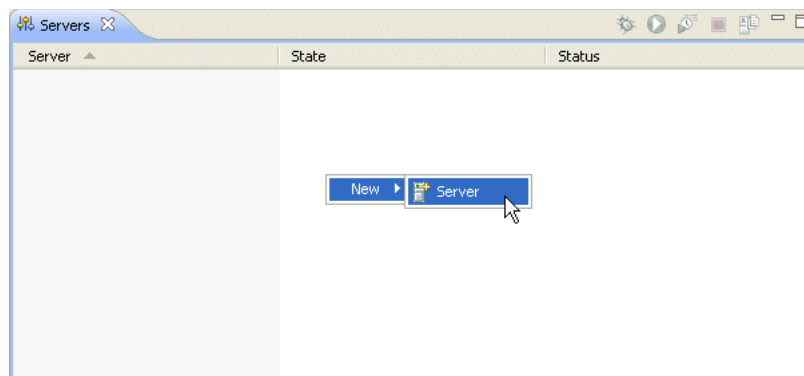
- [Section 5.2.3, "How to Create an Oracle CEP Server Runtime"](#)
- [Section 5.2.1, "How to Create a Local Oracle CEP Server and Server Runtime"](#)

To create a remote Oracle CEP server and server runtime:

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

The Servers view appears as shown in [Figure 5–1](#).

Figure 5–8 Oracle CEP IDE for Eclipse Server View

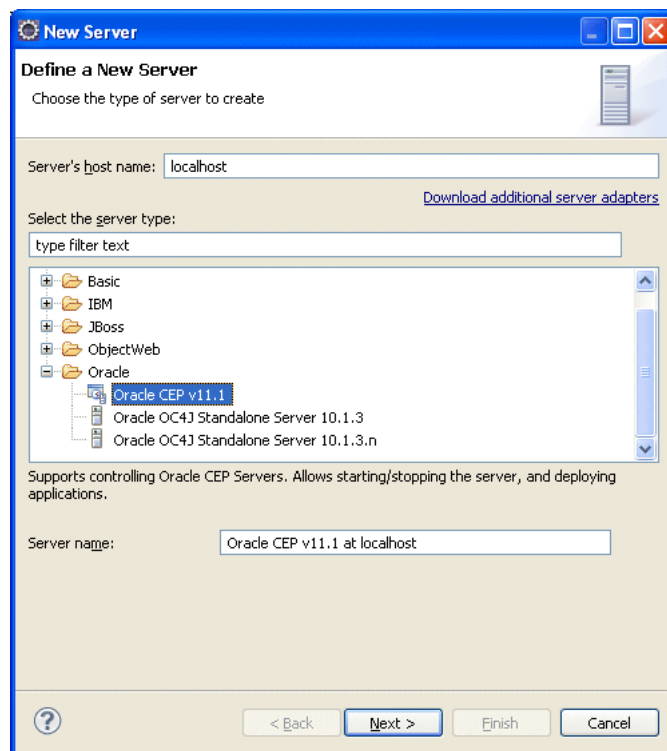


2. Right-click in the **Servers** view pane and select **New > Server**.

3. Consider whether or not server runtimes have been created:
 - a. If this is the first time you have created an Oracle CEP 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 CEP 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 CEP server, there will be no installed server runtimes:

In this case, the New Server: Define New Server dialog appears as [Figure 5–2](#) shows.

Figure 5–9 New Server: Define New Server Dialog (No Installed Runtimes)



Configure the new server as follows:

- a. Configure the dialog as shown in [Table 5–2](#).

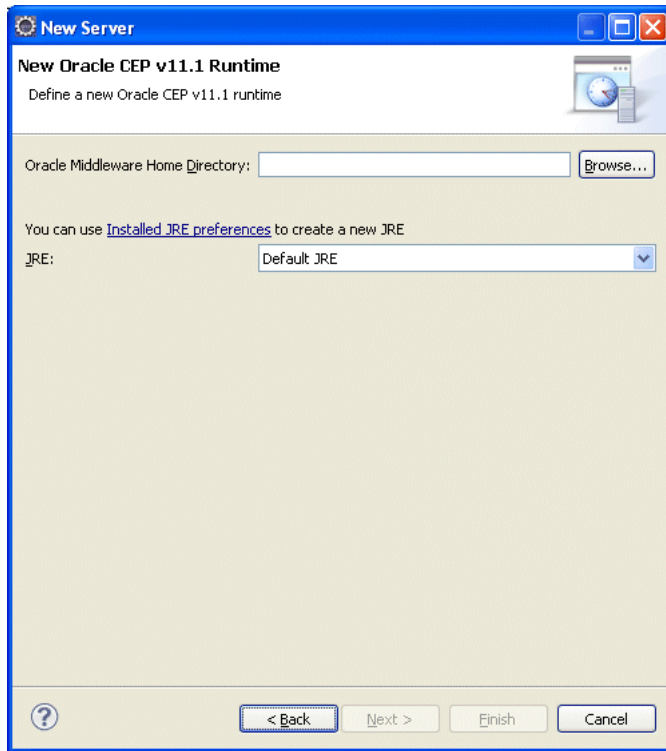
Table 5–6 New Server: Define New Server Dialog (No Installed Runtimes) Attributes

Attribute	Description
Server's host name	The host name of the computer on which you installed Oracle CEP server. For development, this will typically be localhost.
Select the server type	The type of Oracle CEP server. In this example, choose Oracle CEP v11
Server name	The name of this Oracle CEP server. Default: Oracle CEP v11.1 at <i>HOSTNAME</i> Where <i>HOSTNAME</i> is the value you entered in the Server's host name field.

b. Click Next.

The New Server: New Oracle CEP v11 Runtime dialog appears as shown in Figure 5-3.

Figure 5-10 New Server: New Oracle CEP v11.1 Runtime Dialog



c. Configure the dialog as shown in Table 5-3.

Table 5-7 New Server: New Oracle CEP v11 Runtime Dialog Attributes

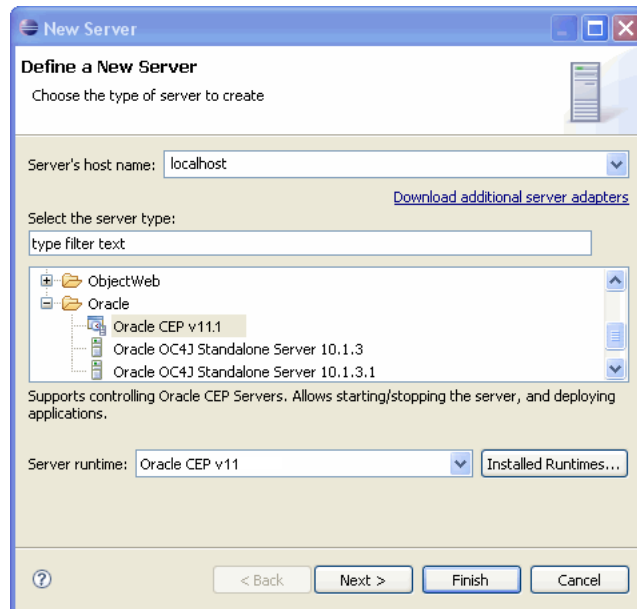
Attribute	Description
Oracle Middleware Home Directory	<p>The fully qualified path to the Oracle CEP server installation directory. This is the same as the "Middleware Home" that was selected when installing the server.</p> <p>When selecting this directory, select the directory that contains the Oracle CEP installation rather than the Oracle CEP 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 CEP installation directory.</p> <p>For more information, see "Oracle Fusion Middleware Directory Structure and Concepts" in the <i>Oracle Complex Event Processing Getting Started</i>.</p>

Table 5–7 (Cont.) New Server: New Oracle CEP v11 Runtime Dialog Attributes

Attribute	Description
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 Installed JRE preferences link to create a new JRE.</p> <p>Be sure to choose a Java 6 JRE.</p> <p>NOTE: The Oracle CEP 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 CEP server, there will be one or more installed server runtimes.
 In this case, the New Server: Define New Server dialog appears as [Figure 5–4](#) shows.

Figure 5–11 New Server: Define New Server (Installed Runtimes) Dialog



Configure the new server as follows:

Table 5–8 New Server: Define New Server (Installed Runtimes) Dialog Attributes

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

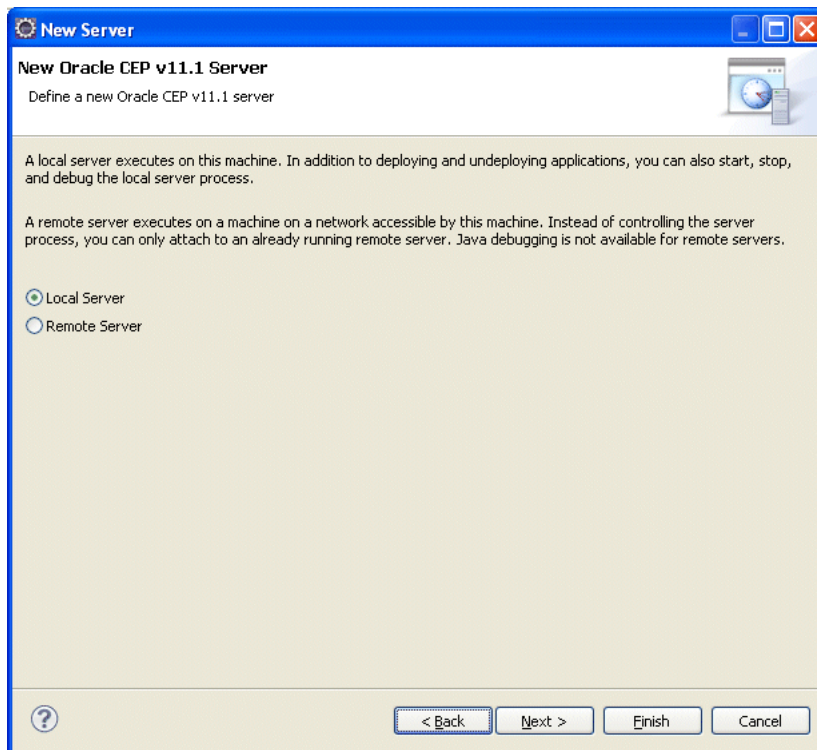
Table 5–8 (Cont.) New Server: Define New Server (Installed Runtimes) Dialog Attributes

Attribute	Description
Server runtime	Select the server runtime from the pull-down menu. To create or edit server runtimes, click Installed Runtimes . For more information, see Section 5.2.3, "How to Create an Oracle CEP Server Runtime" .

6. Click **Next**.

The New Server: New Oracle CEP v11.1 Server dialog appears as [Figure 5–5](#) shows.

Figure 5–12 New Server: New Oracle CEP v11.1 Server



7. Select **Remote Server**.

8. Click **Next**.

The New Server: New Oracle CEP v11 Server dialog appears dialog appears as [Figure 5–13](#) shows.

Figure 5–13 New Server: New Oracle CEP v11 Server Dialog for a Remote Server

9. Configure the dialog as shown in [Table 5–9](#).

Table 5–9 New Server: New Oracle CEP v11 Server Dialog Attributes for a Local Server

Attribute	Description
Remote Server IP Address	The IP address of the remote Oracle CEP server. Default: IP address of localhost.
Remote Server Port	The port you specified in the remote Oracle CEP server <i>DOMAIN_DIR/config/config.xml</i> file that describes your Oracle CEP domain, where <i>DOMAIN_DIR</i> 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><Netio> <Name>NetIO</Name> <Port>9002</Port> </Netio></pre> Default: 9002
User Name	The user name that the Oracle CEP IDE for Eclipse uses to log into the remote server. Default: <code>wlevs</code>
User Password	The password that the Oracle CEP IDE for Eclipse uses to log into the remote server. Default: <code>wlevs</code>

10. Click **Finish**.

5.2.3 How to Create an Oracle CEP Server Runtime

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

You only need to create a runtime explicitly if you have not yet created an Oracle CEP server.

For more information, see:

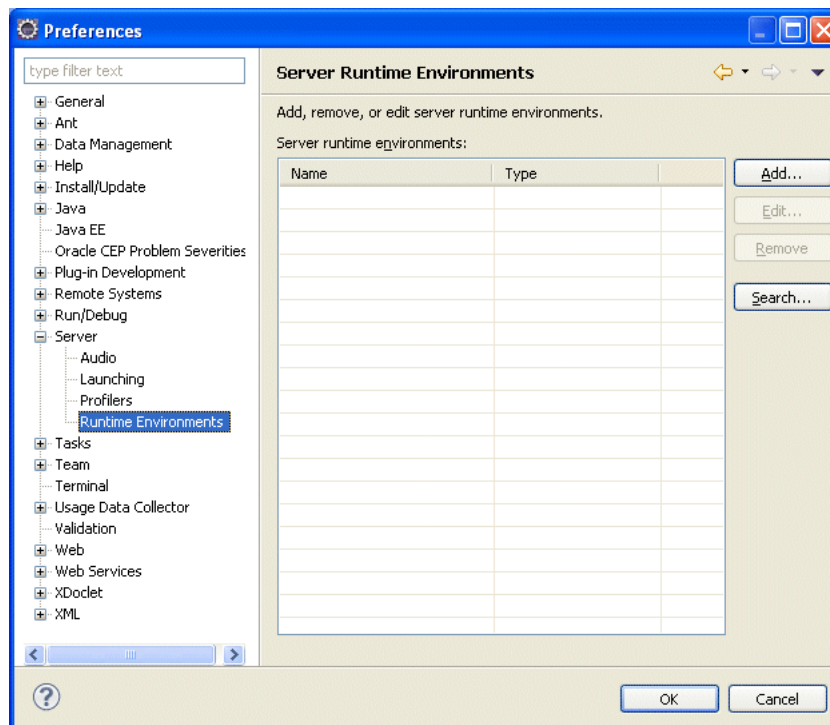
- [Section 5.2.1, "How to Create a Local Oracle CEP Server and Server Runtime"](#)
- [Section 5.2.2, "How to Create a Remote Oracle CEP Server and Server Runtime"](#)

To create an Oracle CEP server runtime:

1. Select **Windows > Preferences**.

The Preferences dialog appears as [Figure 5–14](#) shows.

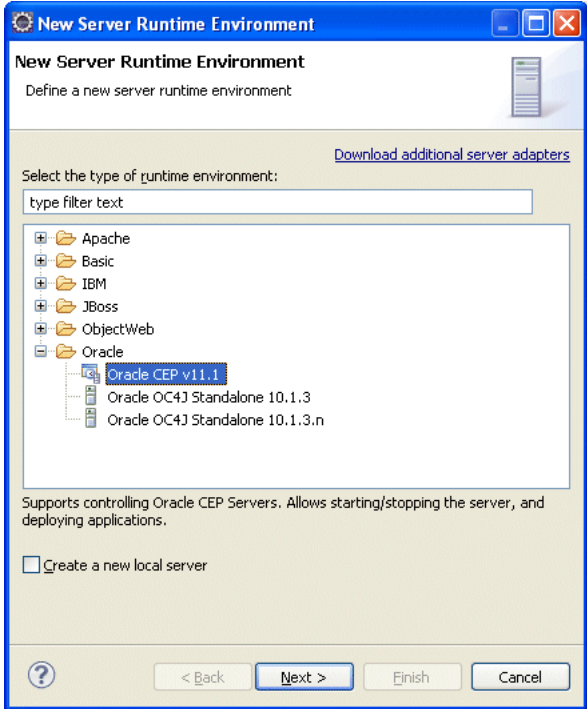
Figure 5–14 Preferences - Server - Installed Runtimes



2. Expand the **Server** option and select **Runtime Environments**.
3. Click **Add**.

The New Server Runtime Environment dialog appears as shown in [Figure 5–15](#).

Figure 5–15 New Server Runtime Environment Dialog



4. Configure the dialog as shown in Table 5–10.

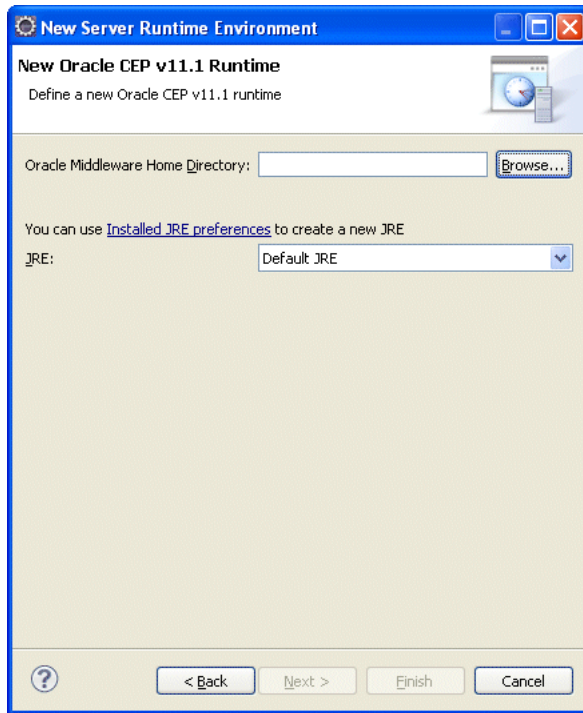
Table 5–10 New Server Runtime Dialog Attributes

Attribute	Description
Select the type of runtime environment	The type of Oracle CEP server. In this example, choose Oracle CEP 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 Section 5.2.1, "How to Create a Local Oracle CEP Server and Server Runtime".

5. Click Next.

The New Server Runtime Environment dialog appears as shown in Figure 5–16.

Figure 5–16 New Server Runtime Environment: New Oracle CEP v11.1 Runtime Dialog



6. Configure the dialog as shown in [Table 5–11](#).

Table 5–11 New Server Runtime Dialog Attributes

Attribute	Description
Oracle Middleware Home Directory	<p>The fully qualified path to the Oracle CEP server installation directory. This is the same as the "Middleware Home" that was selected when installing the server.</p> <p>When selecting this directory, select the directory that contains the Oracle CEP installation rather than the Oracle CEP 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 CEP installation directory.</p> <p>For more information, see "Oracle Fusion Middleware Directory Structure and Concepts" in the <i>Oracle Complex Event Processing Getting Started</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 Installed JRE preferences link to create a new JRE.</p> <p>Be sure to choose either JRockit Real Time or the JRockit JDK installed with your Oracle CEP installation.</p>

7. Click **Finish**.

5.3 Managing Oracle CEP Servers

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

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

- [Section 5.3.1, "How to Start a Local Oracle CEP Server"](#)
- [Section 5.3.2, "How to Stop a Local Oracle CEP Server"](#)
- [Section 5.3.3, "How to Attach to an Existing Local Oracle CEP Server Instance"](#)
- [Section 5.3.4, "How to Attach to an Existing Remote Oracle CEP Server Instance"](#)
- [Section 5.3.5, "How to Detach From an Existing Oracle CEP Server Instance"](#)
- [Section 5.3.6, "How to Deploy an Application to an Oracle CEP Server"](#)
- [Section 5.3.7, "How to Configure Connection and Control Settings for Oracle CEP Server"](#)
- [Section 5.3.8, "How to Configure Domain \(Runtime\) Settings for Oracle CEP Server"](#)
- [Section 5.3.9, "How to Start the Oracle CEP Visualizer from Oracle CEP IDE for Eclipse"](#)

5.3.1 How to Start a Local Oracle CEP Server

After you create a local server, you can start the Oracle CEP server from the Oracle CEP IDE for Eclipse.

You can also start the local Oracle CEP server in debug mode.

Alternatively, you can start the local Oracle CEP server from the command line and attach to it using Oracle CEP IDE for Eclipse.

For more information, see:

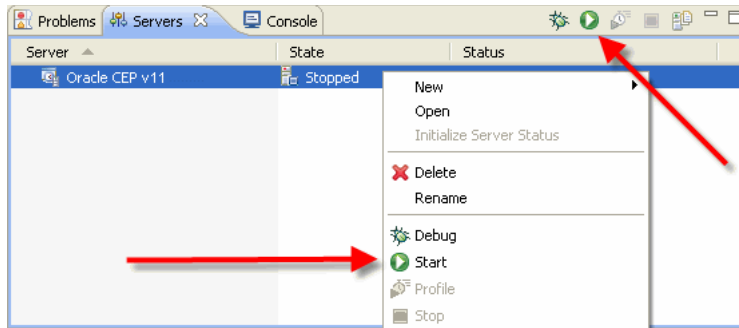
- [Section 5.2.1, "How to Create a Local Oracle CEP Server and Server Runtime"](#)
- [Section 5.4.1, "How to Debug an Oracle CEP Application Running on an Oracle CEP Server"](#)
- [Section 5.3.3, "How to Attach to an Existing Local Oracle CEP Server Instance"](#)

To start a local Oracle CEP server:

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

The Servers view opens as shown in [Figure 5–17](#).

Figure 5–17 Starting an Oracle CEP Server



2. Start the server by choosing one of the following:
 - a. Click the **Start the Server** icon in the Servers view tool bar.
 - b. Right-click a server in the Servers view and select **Start**.

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

5.3.2 How to Stop a Local Oracle CEP Server

After you start a local Oracle CEP server from the Oracle CEP IDE for Eclipse, you can stop the Oracle CEP server from the Oracle CEP IDE for Eclipse.

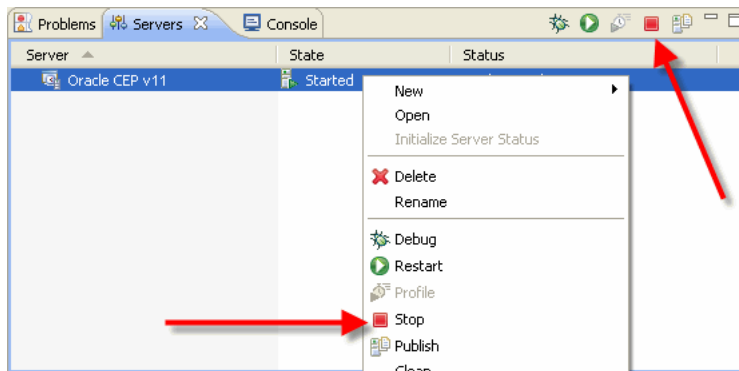
For more information, see [Section 5.3.1, "How to Start a Local Oracle CEP Server"](#).

To stop a local Oracle CEP server:

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

The Servers view opens as shown in [Figure 5–18](#).

Figure 5–18 Stopping an Oracle CEP Server



2. Stop the server by choosing one of the following:
 - a. Click the **Stop the Server** icon in the Servers view tool bar.
 - b. Right-click a server in the Servers view and select **Stop**.

5.3.3 How to Attach to an Existing Local Oracle CEP Server Instance

After you create a local server, you can start the local Oracle CEP server from the command line and attach Oracle CEP IDE for Eclipse to this existing, already running local Oracle CEP server instance.

Alternatively, you can start the local Oracle CEP server directly from within Oracle CEP IDE for Eclipse.

For more information, see:

- [Section 5.2.1, "How to Create a Local Oracle CEP Server and Server Runtime"](#)
- [Section 5.3.1, "How to Start a Local Oracle CEP Server"](#)

To attach to an existing local Oracle CEP server instance:

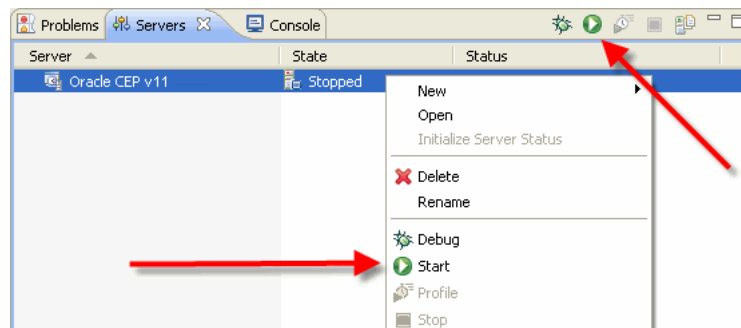
1. Start the Oracle CEP server from the command line.

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

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

The Servers view opens as shown in [Figure 5–17](#).

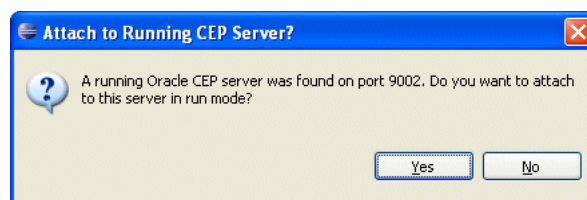
Figure 5–19 Attaching to an Existing Local Oracle CEP Server Instance



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 CEP Server dialog appears as [Figure 5–20](#) shows.

Figure 5–20 Attach to Running CEP Server



4. Click **Yes**.

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

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

5.3.4 How to Attach to an Existing Remote Oracle CEP Server Instance

After you create a remote server, you can start the remote Oracle CEP server from the command line and attach Oracle CEP IDE for Eclipse to this existing, already running remote Oracle CEP server instance.

For more information, see [Section 5.2.2, "How to Create a Remote Oracle CEP Server and Server Runtime"](#).

To attach to an existing remote Oracle CEP server instance:

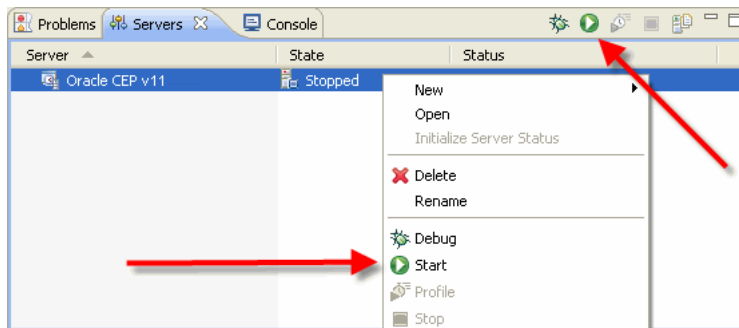
1. Start the remote Oracle CEP server from the command line.

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

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

The Servers view opens as shown in [Figure 5-17](#).

Figure 5-21 Attaching to an Existing Remote Oracle CEP 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 CEP IDE for Eclipse writes one status message to the Console view, reading:

```
[3/23/10 12:32 PM] Attached to remote CEP 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 CEP Visualizer. For more information, see "How to View Console Output" in the *Oracle Complex Event Processing Visualizer User's Guide*.

5.3.5 How to Detach From an Existing Oracle CEP Server Instance

After you attach to an existing, running Oracle CEP server instance, you can detach from the Oracle CEP server and leave it running.

For more information, see:

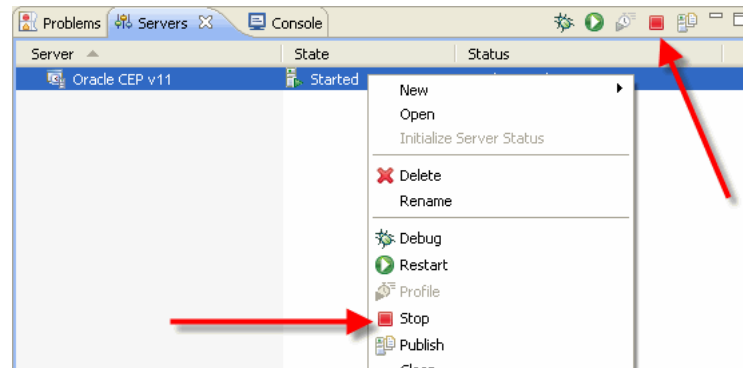
- [Section 5.3.3, "How to Attach to an Existing Local Oracle CEP Server Instance"](#)
- [Section 5.3.4, "How to Attach to an Existing Remote Oracle CEP Server Instance"](#)

To detach from an existing Oracle CEP server instance:

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

The Servers view opens as shown in [Figure 5–18](#).

Figure 5–22 Stopping an Oracle CEP Server



2. Detach from the server by choosing one of the following:
 - a. Click the **Stop the Server** icon in the Servers view tool bar.
 - b. Right-click a server in the Servers view and select **Stop**.

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

If you detach from a remote Oracle CEP server, Oracle CEP IDE for Eclipse writes a log message to the Console view reading:

```
[3/23/10 12:47 PM] Server communication stopped
```

5.3.6 How to Deploy an Application to an Oracle CEP Server

A project in the Oracle CEP IDE for Eclipse is built as an Oracle CEP application, then deployed to the server. To deploy an application, a server must first be defined. 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:

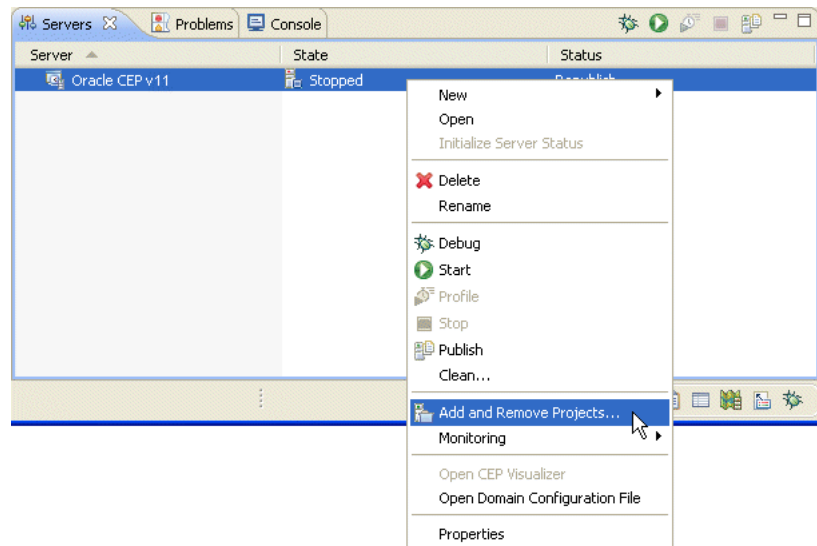
- [Section 5.2.1, "How to Create a Local Oracle CEP Server and Server Runtime"](#)
- [Section 5.2.2, "How to Create a Remote Oracle CEP Server and Server Runtime"](#)

To deploy an application to an Oracle CEP server:

1. Create an Oracle CEP project (see [Section 4.2, "Creating Oracle CEP Projects"](#)).
2. Create a server (see [Section 5.2.1, "How to Create a Local Oracle CEP Server and Server Runtime"](#)).
3. Select **Window > Show Views > Servers**.

The Servers view opens as shown in [Figure 5–23](#).

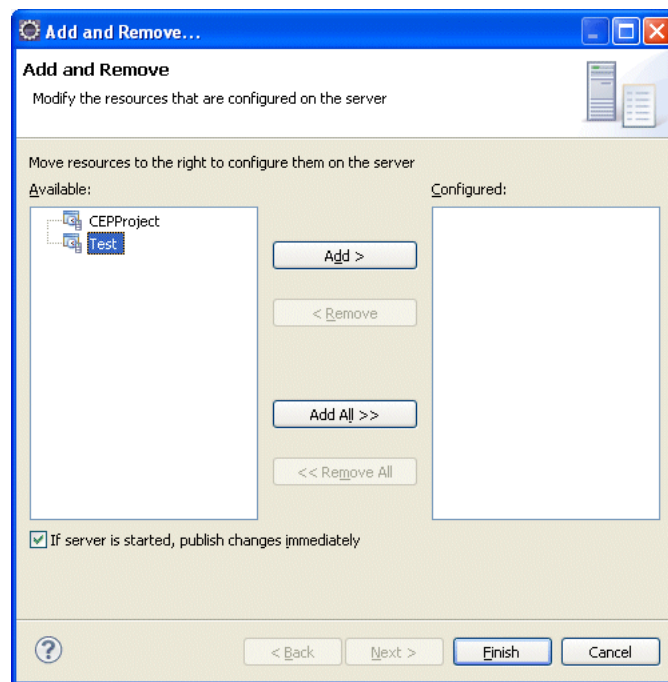
Figure 5–23 Adding a Project to an Oracle CEP Server



4. Right-click the server and select **Add and Remove**.

The Add and Remove dialog appears as shown in [Figure 5–24](#).

Figure 5–24 Add and Remove Dialog



5. Configure the dialog as [Table 5–12](#) shows.

Table 5–12 Add and Remove Dialog Attributes

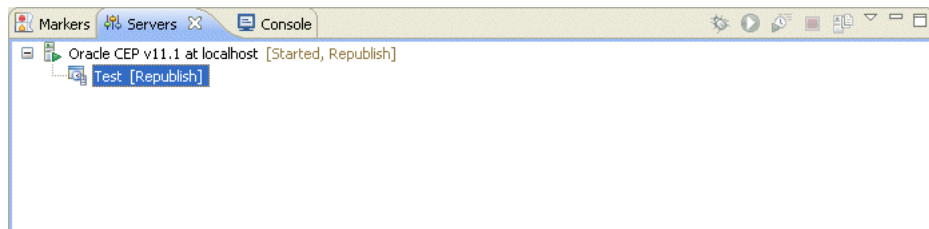
Attribute	Description
Available	Select one or more projects from this list and click Add or Add All to move them into the Configured list.

Table 5–12 (Cont.) Add and Remove Dialog Attributes

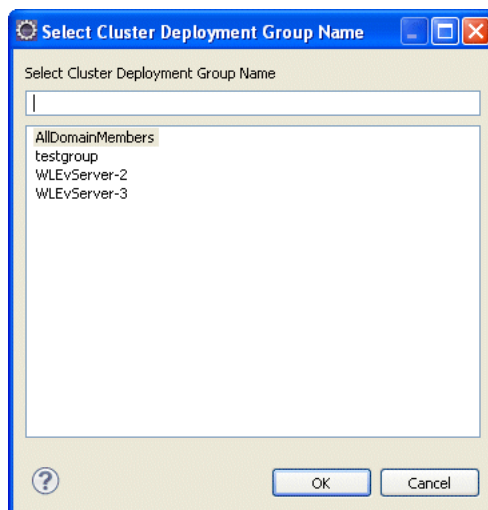
Attribute	Description
Configured	Select one or more projects from this list and click Remove or Remove All to move them into the Available list.
If server is started, publish changes immediately.	Check this option to immediately publish projects that you modify. Applicable only if the server is already 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 5–25](#).

Figure 5–25 Server View After Adding a Project

7. To deploy (publish) the application to the Oracle CEP server, right-click the added application and select **Force Publish**.
- If the Oracle CEP server is part of a standalone-server, domain the application is deployed.
 - If the Oracle CEP server is part of a multi-server domain, the Select Cluster Deployment Group Name dialog appears as [Figure 5–26](#) shows.

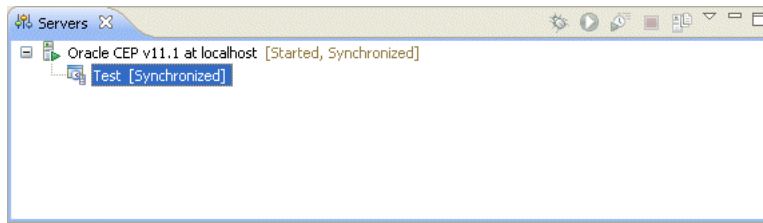
Figure 5–26 Select Cluster Deployment Group Name Dialog

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 Complex Event Processing Administrator's Guide*.

Once an application is deployed (published), it will show as a child of the server in the Servers view as shown in [Figure 5-27](#).

Figure 5-27 Server View After Deploying (Publishing) a Project



5.3.7 How to Configure Connection and Control Settings for Oracle CEP 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 CEP IDE for Eclipse uses to communicate with the Oracle CEP server.

For more information, see:

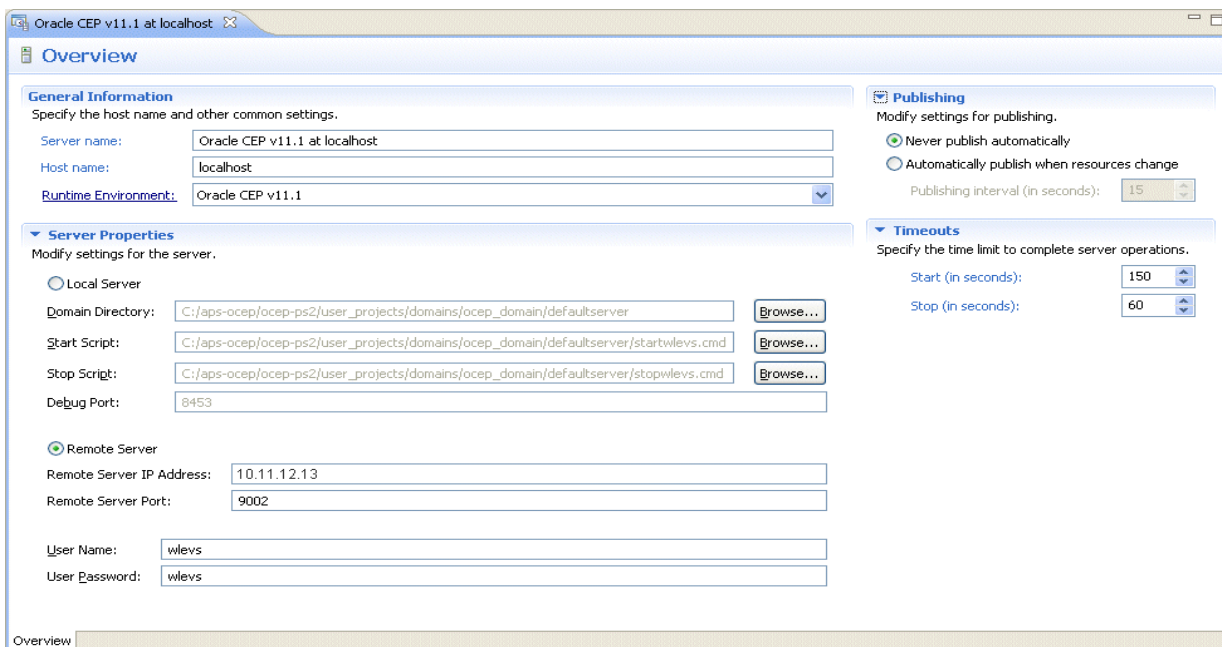
- [Section 5.2.1, "How to Create a Local Oracle CEP Server and Server Runtime"](#)
- [Section 5.3.8, "How to Configure Domain \(Runtime\) Settings for Oracle CEP Server"](#)

To configure connection and control settings for Oracle CEP server:

1. Select **Window > Show Views > Servers**.
2. Double-click a server in the Servers view.

The Server Overview editor opens as shown in [Figure 5-28](#).

Figure 5-28 Server Overview Editor



3. Configure the Server Overview editor as shown in [Table 5-13](#).

Table 5–13 Server Overview Editor Attributes

Attribute	Description
Server Name	The name of this server. Only used within the Oracle CEP IDE for Eclipse as a useful identifier. For more information, see Section 5.2.1, "How to Create a Local Oracle CEP Server and Server Runtime" .
Host Name	The name of the host on which this server is installed. For more information, see Section 5.2.1, "How to Create a Local Oracle CEP Server and Server Runtime" .
Runtime Environment	The current installed runtime selected for this server. Select a new runtime from the pull down menu or click the Edit link to modify the configuration of the selected runtime. For more information, see Section 5.2.1, "How to Create a Local Oracle CEP Server and Server Runtime" .
Domain Directory ¹	The fully qualified path to the directory that contains the domain for this server. Click Browse to choose the directory. Default: <code>ORACLE-CEP-HOME\user_projects\domains\ocep_domain\defaultserver</code> .
Start Script ¹	The script that Oracle CEP IDE for Eclipse uses to start the Oracle CEP server. Click Browse to choose the start script. Default on UNIX: <code>ORACLE-CEP-HOME\user_projects\domains\ocep_domain\defaultserver\stopwlevs.sh</code> Default on Windows: <code>ORACLE-CEP-HOME\user_projects\domains\ocep_domain\defaultserver\stopwlevs.cmd</code>
Stop Script ¹	The script that Oracle CEP IDE for Eclipse uses to stop the Oracle CEP server. Click Browse to choose the stop script. Default on UNIX: <code>ORACLE-CEP-HOME\user_projects\domains\ocep_domain\defaultserver\startwlevs.sh</code> Default on Windows: <code>ORACLE-CEP-HOME\user_projects\domains\ocep_domain\defaultserver\startwlevs.cmd</code>
Debug Port ¹	The Oracle CEP server port that Oracle CEP IDE for Eclipse connects to when debugging the Oracle CEP server. Default: 8453.
Remote Server IP Address ²	The IP address of the remote Oracle CEP server. Default: IP address of localhost.

Table 5–13 (Cont.) Server Overview Editor Attributes

Attribute	Description
Remote Server Port ²	<p>The port you specified in the remote Oracle CEP server <code>DOMAIN_DIR/config/config.xml</code> file that describes your Oracle CEP 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><Netio> <Name>NetIO</Name> <Port>9002</Port> </Netio></pre> <p>Default: 9002</p>
User Name ²	<p>The user name that the Oracle CEP IDE for Eclipse uses to log into the remote server.</p> <p>Default: <code>wlevs</code></p>
User Password ²	<p>The password that the Oracle CEP 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 CEP server.</p> <p>Select Never publish automatically to disable automatic publishing.</p> <p>Select Override default settings to override the default automatic publishing interval. Enter a new publishing interval (in seconds).</p> <p>Default: <code>Never publish automatically</code>.</p>
Timeouts	<p>Enter a positive, integer number of seconds in the Start (in seconds) field to specify the time in which the Oracle CEP server must start.</p> <p>Default: 150 seconds.</p> <p>Enter a positive, integer number of seconds in the Stop (in seconds) field to specify the time in which the Oracle CEP server must start.</p> <p>Default: 60 seconds.</p>

¹ Click **Local Server** to modify. Applies to both a local server and the runtime of a remote server.

² Click **Remote Server** to modify. Applies only to a remote server.

4. Select **File > Save**.
5. Close the Server Overview editor.

5.3.8 How to Configure Domain (Runtime) Settings for Oracle CEP Server

After you create a server, you can use the Oracle CEP IDE for Eclipse to configure Oracle CEP server domain (runtime) settings in the Oracle CEP server `config.xml` file.

Recall that a local Oracle CEP server is one in which both the server and server runtime are on the same host and a remote Oracle CEP 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 CEP IDE for Eclipse).

For both local and remote Oracle CEP servers, when you configure domain (runtime) settings, you are modifying only the Oracle CEP server `config.xml` on the local host.

You can also use the Oracle CEP IDE for Eclipse to configure all the important server connection and control settings that Oracle CEP IDE for Eclipse uses to communicate with the Oracle CEP server.

Any changes you make to the Oracle CEP server `config.xml` file for a running Oracle CEP server are not read by the Oracle CEP server until you restart it.

If you make changes to the Oracle CEP server `config.xml` file for a running Oracle CEP server using the Oracle CEP Visualizer, the changes apply to the running Oracle CEP server as soon as you save them. The Oracle CEP Visualizer updates the Oracle CEP 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 CEP server `config.xml` file by manually editing this file, and you then make further changes using the Oracle CEP Visualizer, your manual edits will be overwritten by the Oracle CEP Visualizer.

To avoid this, when you manually edit the Oracle CEP server `config.xml` file, always stop and start the Oracle CEP server to read those changes into the runtime configuration and then use the Oracle CEP Visualizer to make further changes.

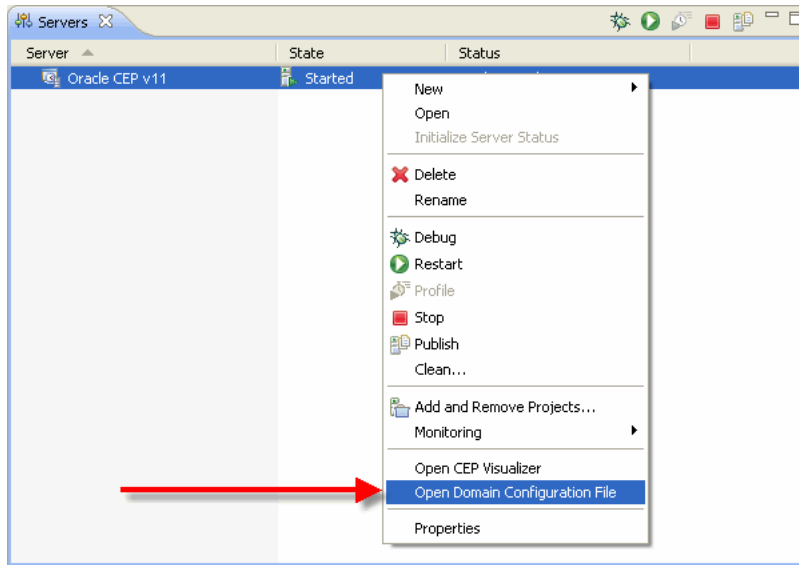
For more information, see:

- [Section 5.2.1, "How to Create a Local Oracle CEP Server and Server Runtime"](#)
- [Section 5.2.2, "How to Create a Remote Oracle CEP Server and Server Runtime"](#)
- [Section 5.3.7, "How to Configure Connection and Control Settings for Oracle CEP Server"](#)
- [Section 5.3.9, "How to Start the Oracle CEP Visualizer from Oracle CEP IDE for Eclipse"](#)
- "Understanding Oracle CEP Server Configuration" in the *Oracle Complex Event Processing Administrator's Guide*

To configure domain (runtime) settings for Oracle CEP server:

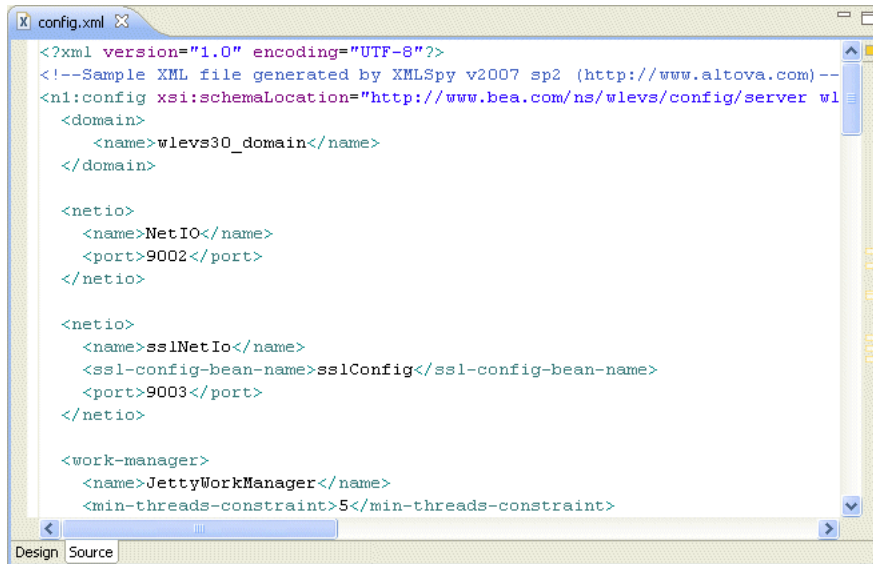
1. Select **Window > Show Views > Servers**.
2. Right-click a server in the Servers view and select **Open Domain Configuration File** as shown in [Figure 5-29](#).

Figure 5–29 Editing the Domain Configuration File



The Oracle CEP server domain configuration file `config.xml` opens as shown in Figure 5–30.

Figure 5–30 Oracle CEP Domain Configuration File `config.xml`



3. Edit the domain configuration file as required.
4. Select **File > Save**.
5. Close the domain configuration file.

5.3.9 How to Start the Oracle CEP Visualizer from Oracle CEP IDE for Eclipse

After you create a server, you can start the Oracle CEP Visualizer from the Oracle CEP IDE for Eclipse.

The Oracle CEP Visualizer is the administration console for a running Oracle CEP server. For more information, see the Oracle Complex Event Processing Visualizer User's Guide.

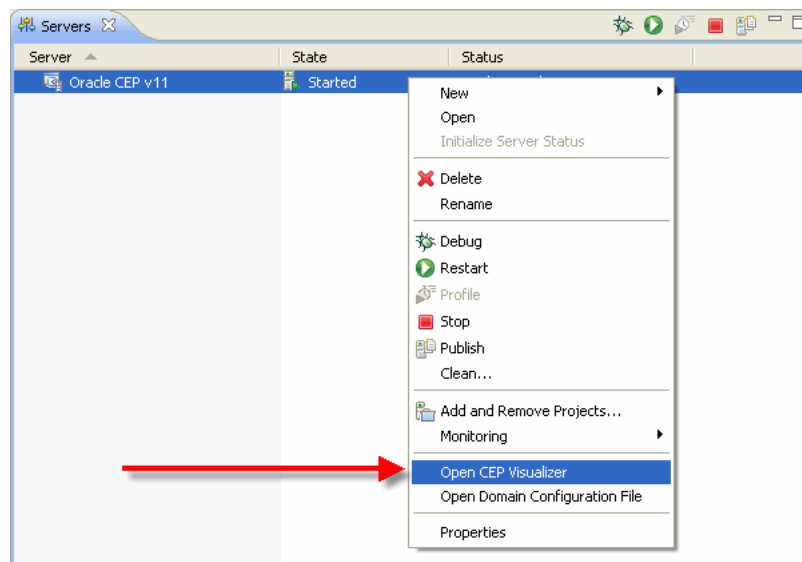
For more information, see [Section 5.2.1, "How to Create a Local Oracle CEP Server and Server Runtime"](#).

Note: If you use the Oracle CEP Visualizer to make changes to the Oracle CEP server `config.xml` (for example, editing a data source), you may overwrite `config.xml` file changes made manually. For more information, see [Section 5.3.8, "How to Configure Domain \(Runtime\) Settings for Oracle CEP Server"](#).

To start the Oracle CEP Visualizer from Oracle CEP IDE for Eclipse:

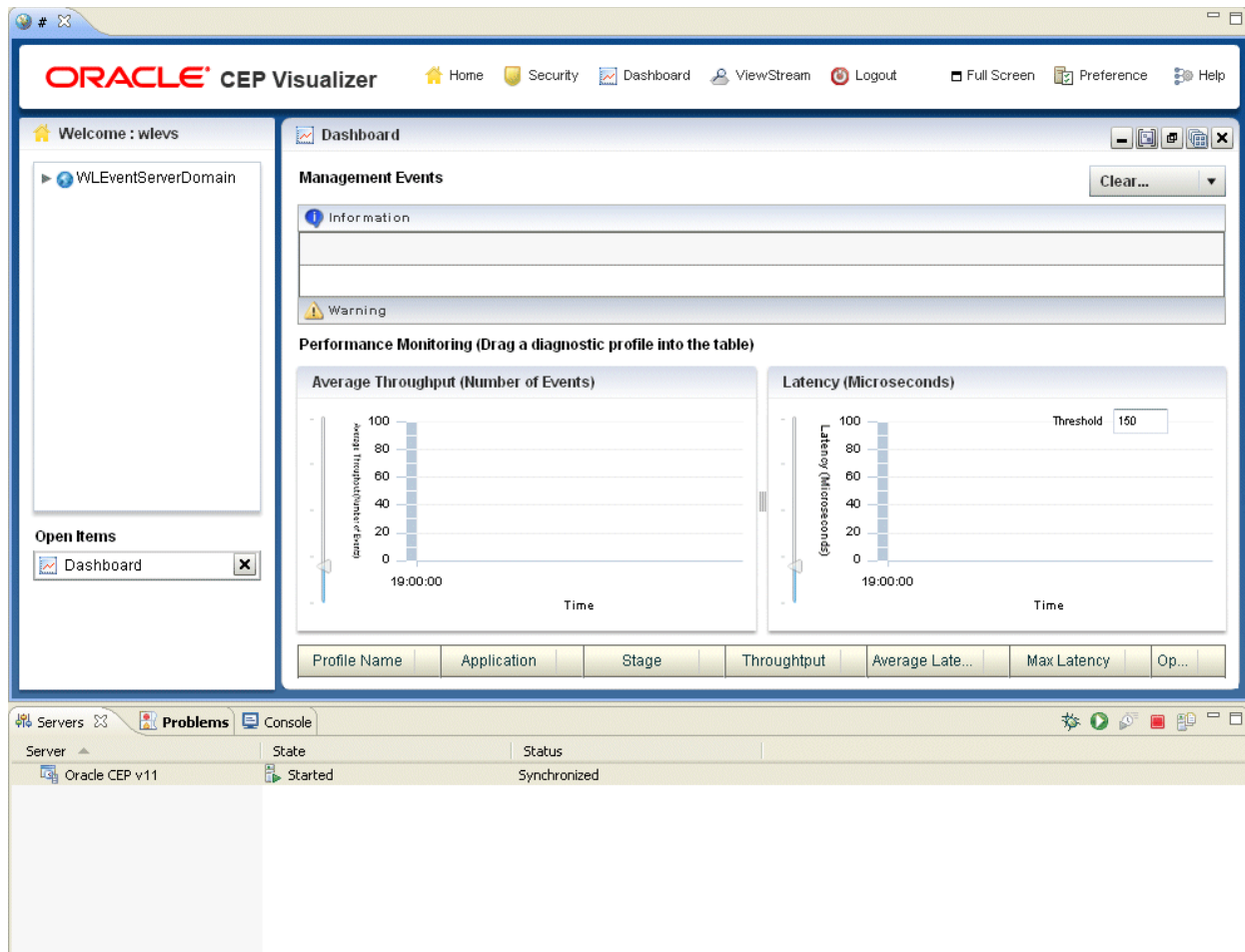
1. Start the server (see [Section 5.3.1, "How to Start a Local Oracle CEP Server"](#)).
2. Right-click the running server in the Servers view and select **Open CEP Visualizer** as shown in [Figure 5–31](#).

Figure 5–31 Opening the Oracle CEP Visualizer



The Oracle CEP Visualizer opens as shown in [Figure 5–32](#).

Figure 5–32 Oracle CEP Visualizer



- Use the Oracle CEP Visualizer as the *Oracle Complex Event Processing Visualizer User's Guide* describes.

5.4 Debugging an Oracle CEP Application Running on an Oracle CEP Server

Because Oracle CEP applications are Java applications, standard Java debugging tools including those provided in Eclipse can be used with these applications.

This section describes:

- Section 5.4.1, "How to Debug an Oracle CEP Application Running on an Oracle CEP Server"

You can also use the load generator and csvgen adapter to simulate data feeds for testing. For more information, see [Chapter 25, "Testing Applications With the Load Generator and csvgen Adapter"](#).

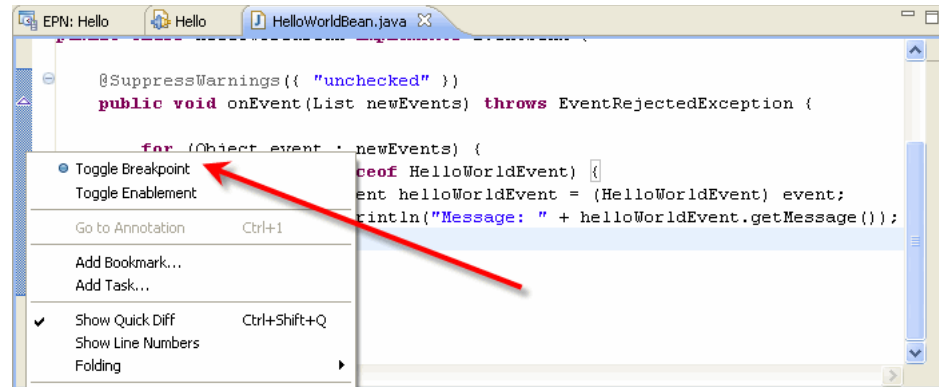
5.4.1 How to Debug an Oracle CEP Application Running on an Oracle CEP Server

This section describes how to debug an Oracle CEP application running on an Oracle CEP server.

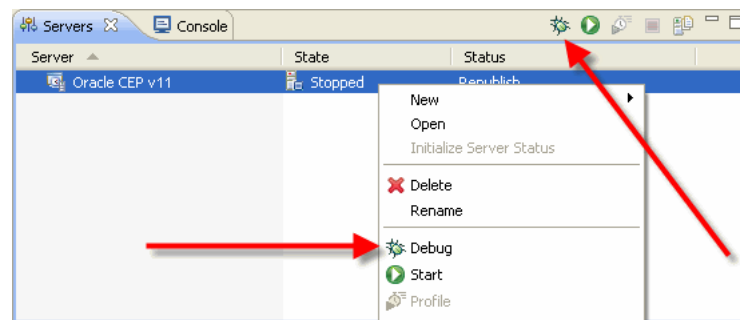
To debug an Oracle CEP application running on an Oracle CEP server:

1. Set a breakpoint in the Java code you wish to debug.

In this case, set the breakpoint by right-clicking in the gutter of the editor and selecting **Toggle Breakpoint** as [Figure 5–33](#) shows.

Figure 5–33 *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 5–34](#):
 - 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 5–34 *Starting the Oracle CEP Server in Debug Mode*

4. The server will start, and when it gets to your breakpoint the thread will stop.
If the Oracle CEP IDE for Eclipse does not automatically switch to the Debug perspective, switch to that perspective by selecting **Window > Open Perspective > Other** and selecting the **Debug** option from the list of perspective.
5. Debug your application using the Debug perspective.

Note: In some cases you may get a dialog box warning that it could not install a breakpoint because of missing line number information. This dialog comes from the core Eclipse debugger and is normally a harmless issue with Oracle CEP Service Engine applications. Simply check the **Don't Tell Me Again** checkbox and continue debugging.

6. When you are finished you can stop the server as usual (see [Section 5.3.2, "How to Stop a Local Oracle CEP Server"](#)).

Oracle CEP IDE for Eclipse and the Event Processing Network

The Oracle CEP Event Processing Network (EPN) is the central concept in an Oracle CEP application. It is the primary point where application components are wired together. Using Oracle CEP IDE for Eclipse, you can use an EPN Editor that provides a graphical view of the EPN and offers visualization and navigation features to help you build Oracle CEP applications.

This section describes how to use the editor and the information it displays, including:

- [Section 6.1, "Opening the EPN Editor"](#)
- [Section 6.2, "EPN Editor Overview"](#)
- [Section 6.3, "Navigating the EPN Editor"](#)
- [Section 6.4, "Using the EPN Editor"](#)

6.1 Opening the EPN Editor

You can open the EPN Editor from either the project folder or a context or configuration file of an Oracle CEP application.

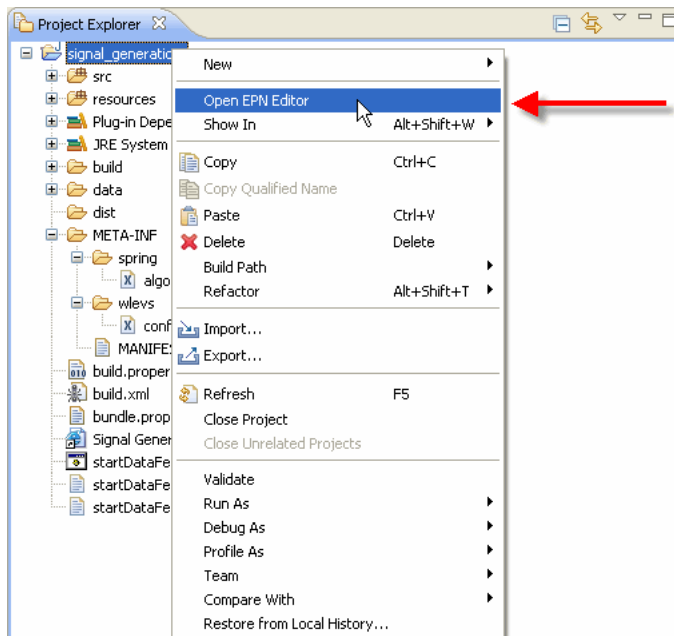
6.1.1 How to Open the EPN Editor from a Project Folder

You can open the EPN Editor from the Eclipse project folder of an Oracle CEP application. Alternatively, you can open the EPN Editor from a context or configuration file (see [Section 6.1.2, "How to Open the EPN Editor from a Context or Configuration File"](#)).

To open the EPN Editor from a project:

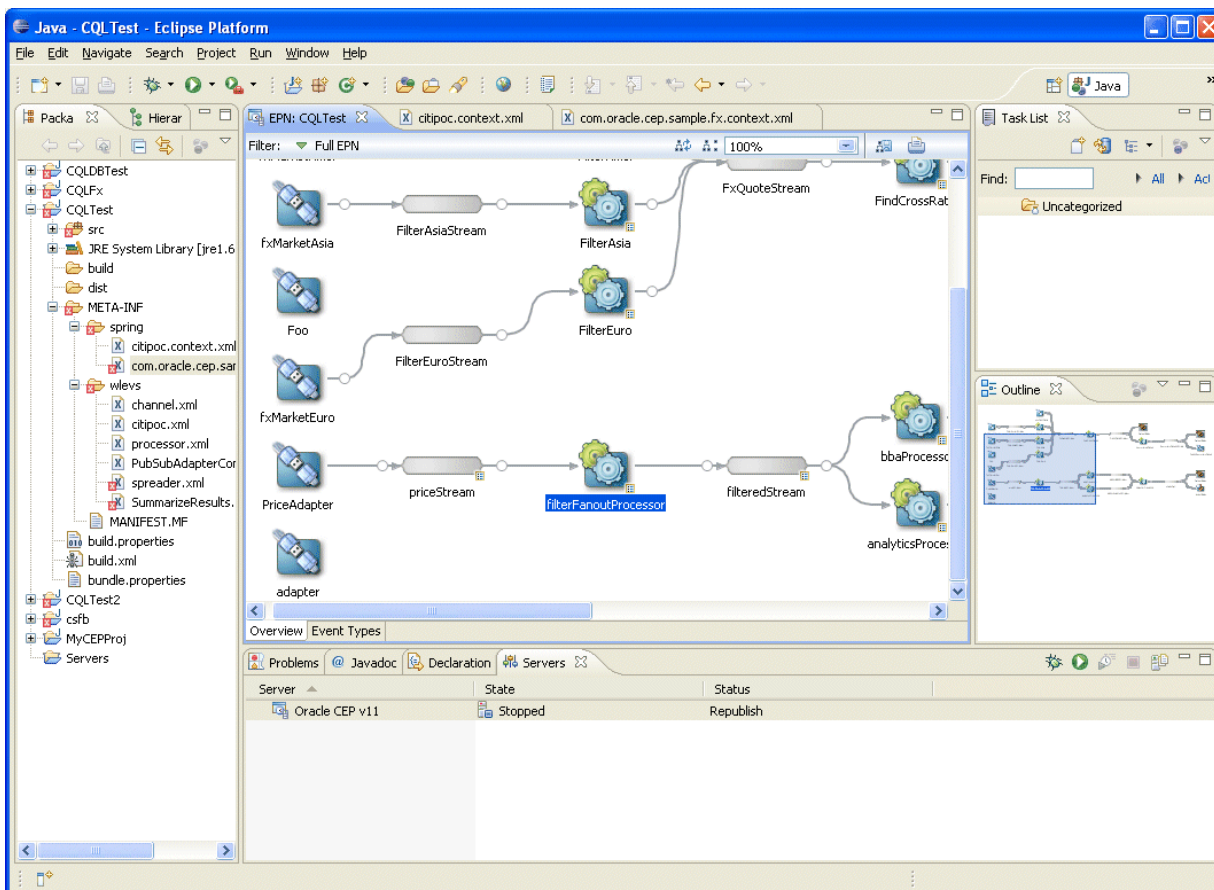
1. Launch the Oracle CEP IDE for Eclipse.
2. Open your Oracle CEP project in the Project Explorer.
3. Right-click the project folder and select **Open EPN Editor** as [Figure 6-1](#) shows.

Figure 6–1 Opening the EPN Editor from a Project



The EPN Editor opens in a tab named `EPN : PROJECT-NAME`, where `PROJECT-NAME` is the name of your Oracle CEP project, as [Figure 6–2](#) shows.

Figure 6–2 EPN Editor



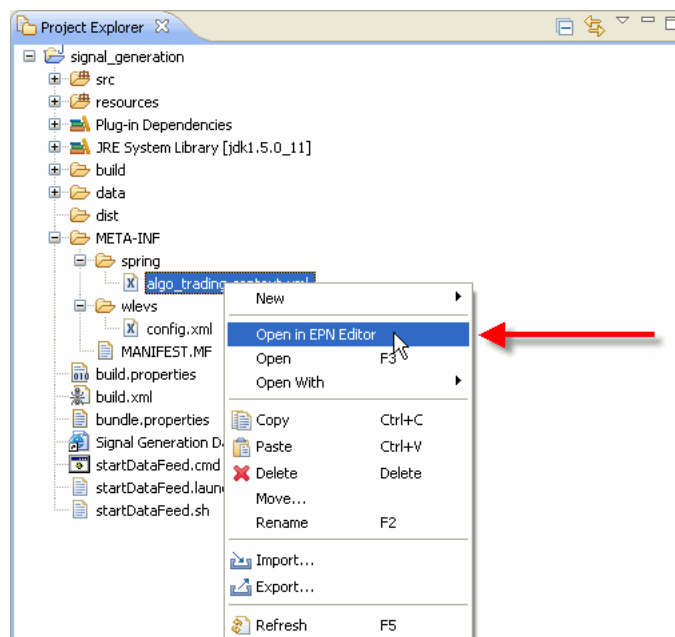
6.1.2 How to Open the EPN Editor from a Context or Configuration File

You can open the EPN Editor from a Spring context file or an Oracle CEP server configuration file in an Oracle CEP application. Alternatively, you can open the EPN Editor from a context or configuration file (see [Section 6.1.1, "How to Open the EPN Editor from a Project Folder"](#))

To open the EPN Editor from a context or configuration file:

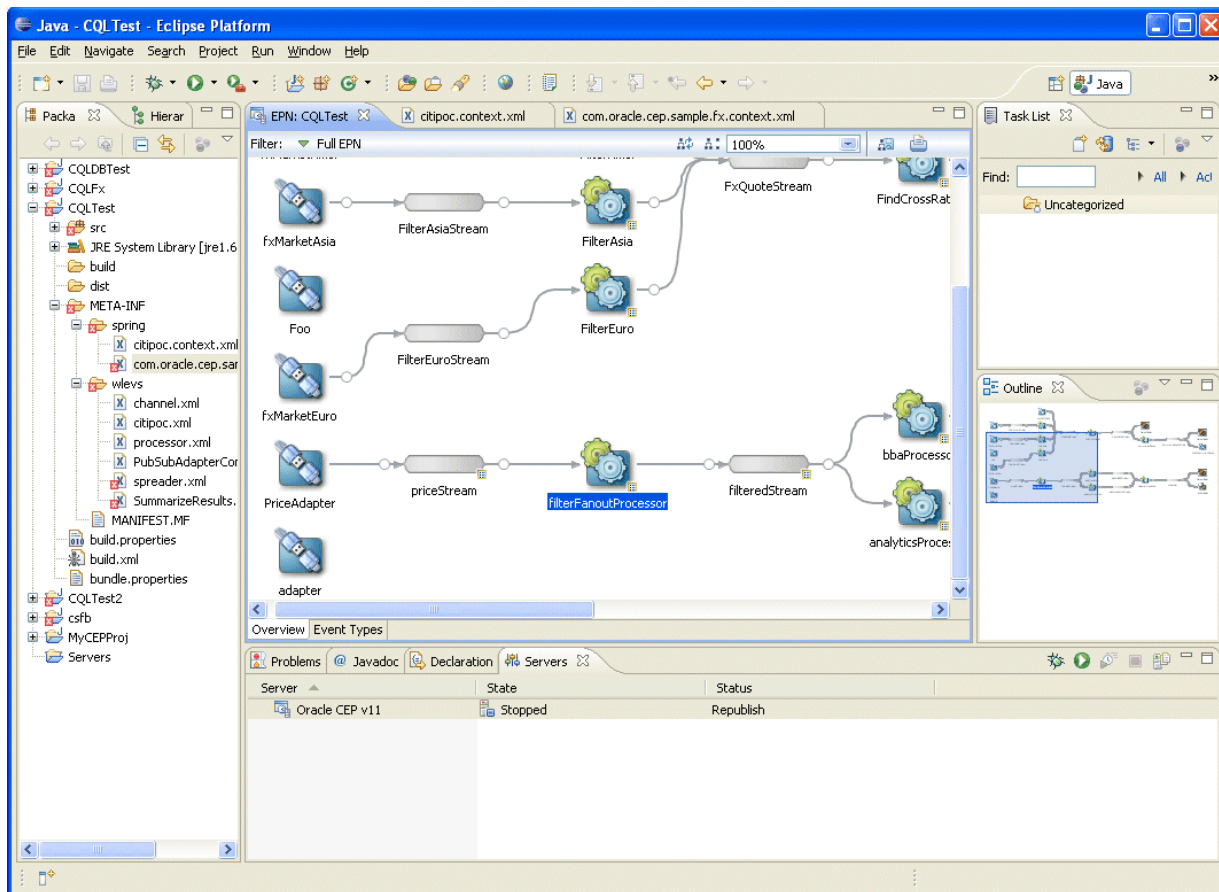
1. Launch the Oracle CEP IDE for Eclipse.
2. Open your Oracle CEP project in the Project Explorer.
3. Right-click a context or configuration file and select **Open in EPN Editor** as [Figure 6–3](#) shows.

Figure 6–3 Opening the EPN Editor from a Context or Configuration File



The EPN Editor opens in a tab named `EPN: PROJECT-NAME`, where `PROJECT-NAME` is the name of your Oracle CEP project, as [Figure 6–4](#) shows.

Figure 6–4 EPN Editor



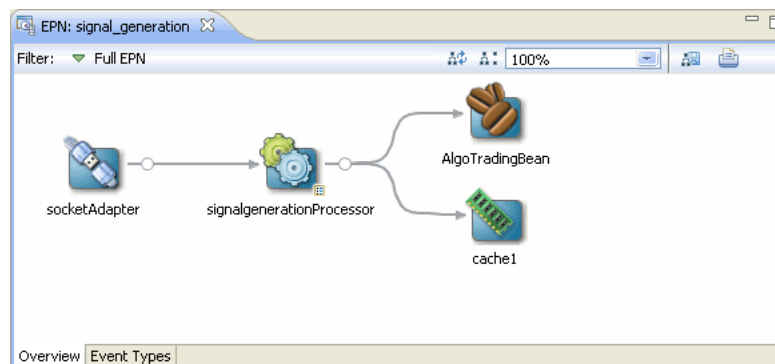
6.2 EPN Editor Overview

This section describes the main controls you use to manage the EPN view and how the EPN Editor displays Oracle CEP application information, including:

- [Section 6.2.1, "Flow Representation"](#)
- [Section 6.2.2, "Filtering"](#)
- [Section 6.2.3, "Zooming"](#)
- [Section 6.2.4, "Layout"](#)
- [Section 6.2.5, "Showing and Hiding Unconnected Beans"](#)
- [Section 6.2.6, "Printing and Exporting to an Image"](#)
- [Section 6.2.7, "Configuration Badging"](#)
- [Section 6.2.8, "Link Specification Location Indicator"](#)
- [Section 6.2.9, "Nested Stages"](#)
- [Section 6.2.10, "Event Type Repository Editor"](#)

6.2.1 Flow Representation

The primary display in the editor is of the flow inside the application as [Figure 6–5](#) shows.

Figure 6–5 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:

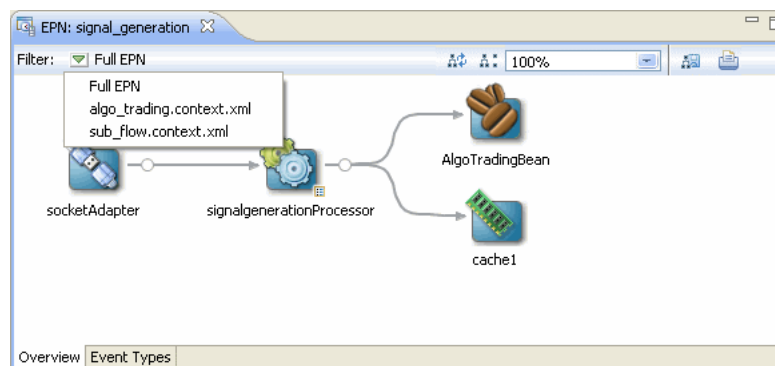
- [Section 6.2.7, "Configuration Badging"](#)
- [Section 6.2.8, "Link Specification Location Indicator"](#)

6.2.2 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 CEP 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 6–6](#) shows.

Figure 6–6 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.

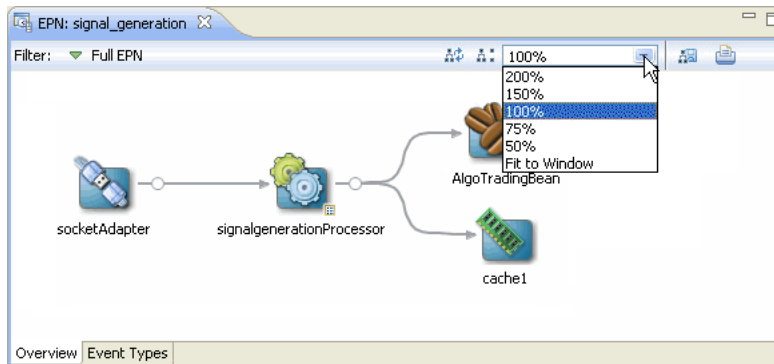
If the assembly file the EPN Editor edits is not open in an Eclipse source editor, then the edits are immediately applied to the file on disk.

For more information, see [Section 4.3, "Creating EPN Assembly Files"](#).

6.2.3 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 6–7](#) shows. To fit the EPN into the current EPN Editor window, select **Fit to Window**.

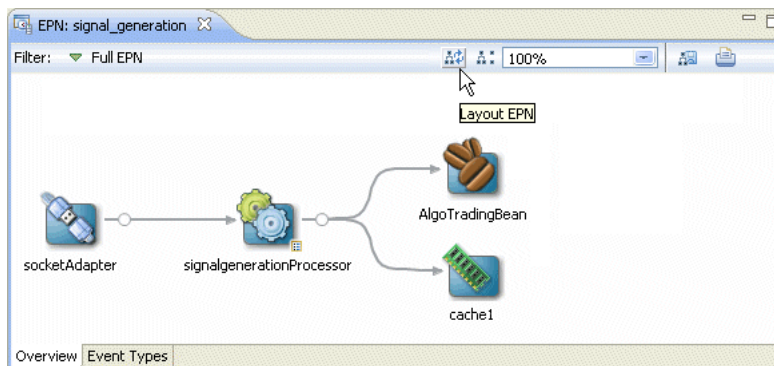
Figure 6–7 Zoom Level



6.2.4 Layout

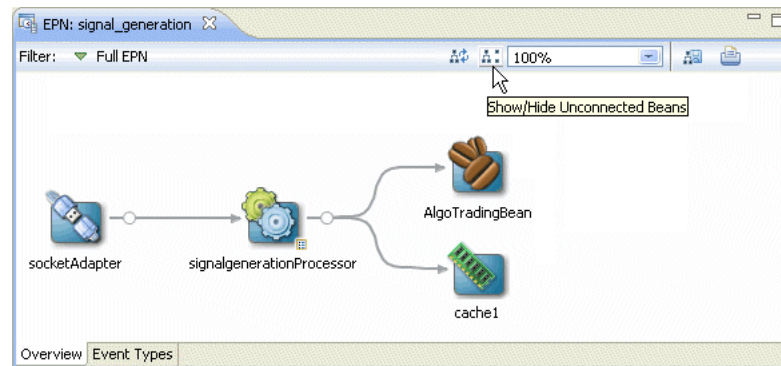
You can optimize and simplify the EPN layout by clicking **Layout EPN** as [Figure 6–8](#) shows.

Figure 6–8 Optimize Layout



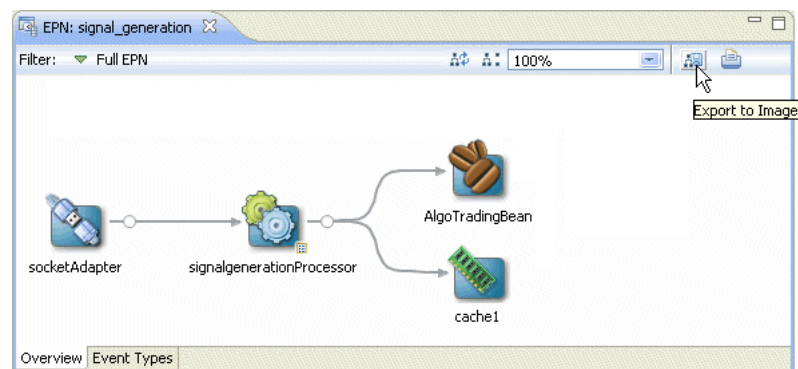
6.2.5 Showing and Hiding Unconnected Beans

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 6–9](#) shows. For more information, see [Section 6.4.3, "Laying Out Nodes"](#).

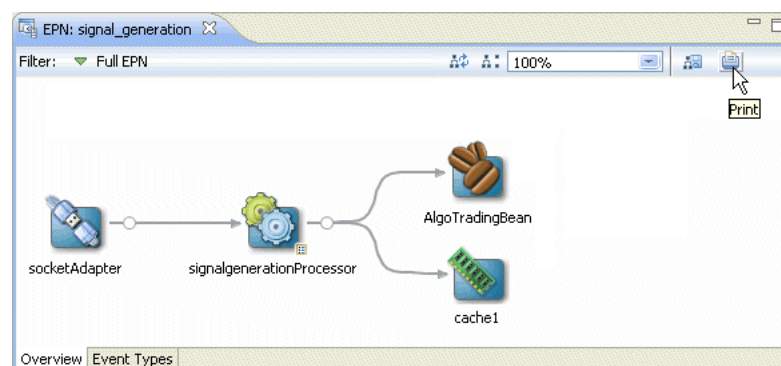
Figure 6–9 Show/Hide Unconnected Beans

6.2.6 Printing and Exporting to an Image

You can export the EPN Editor view to an image file by clicking **Export to Image** as [Figure 6–10](#) shows. You can export the image as a .bmp, .gif, .jpg, or .png file.

Figure 6–10 Exporting the EPN as an Image File

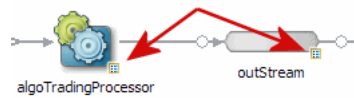
You can print the EPN Editor view by clicking **Print** as [Figure 6–11](#) shows.

Figure 6–11 Printing the EPN

6.2.7 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 6–12](#) shows.

Figure 6–12 Configuration Badging



Nodes with this badge will also have the **Go To Configuration Source** context menu item.

6.2.8 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 6–13 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 6–14 shows the corresponding assembly file.

Figure 6–13 Link Source



Figure 6–14 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 6–15 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 6–16 shows the corresponding assembly file.

Figure 6–15 Link Listener



Figure 6–16 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>

```

6.2.9 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 6.4.2, "Connecting Nodes"](#).

Consider the EPN that [Figure 6–17](#) shows. [Example 6–1](#) 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 6–17 EPN With Nested Bean**Example 6–1 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>

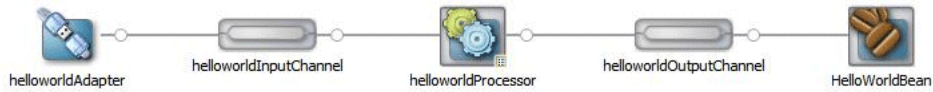
<wlevs:processor id="helloworldProcessor" />

<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 6–18](#) shows. [Example 6–2](#) 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 6–18 EPN With all Nodes Nested



Example 6–2 Assembly Source for EPN With all Nodes Nested

```

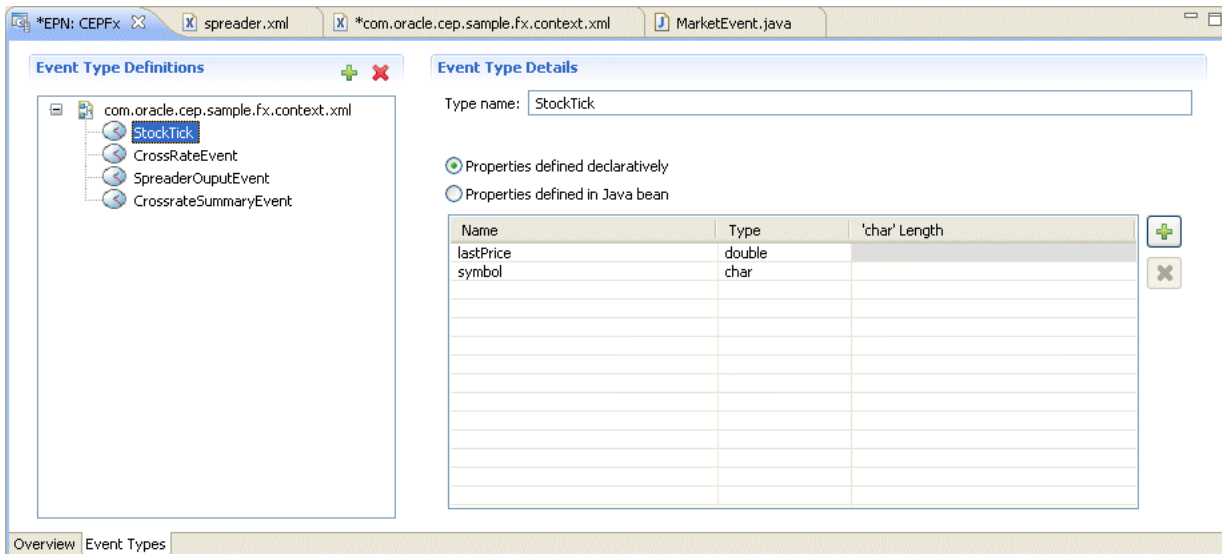
<wlevs:adapter id="helloworldAdapter" class="com.bea.wlevs.adapter.example.helloworld.HelloWorldAdapter" >
  <wlevs:instance-property name="message" value="HelloWorld - the current time is:"/>
  <wlevs:listener>
    <wlevs:channel id="helloworldInputChannel" event-type="HelloWorldEvent" >
      <wlevs:listener>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

6.2.10 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 6–19](#) shows.

Figure 6–19 Event Type Repository Editor



For more information, see:

- [Section 2.2.1, "How to Create an Oracle CEP Event Type as a JavaBean Using the Event Type Repository Editor"](#)
- [Section 2.3.1, "How to Create an Oracle CEP Event Type as a Tuple Using the Event Type Repository Editor"](#)

For information on the other types of events you can create, see [Section 1.1.2, "Oracle CEP Event Types"](#).

6.3 Navigating the EPN Editor

Because the EPN Editor has a view of the whole project it is a natural place from which to navigate to the various artifacts that make up an Oracle CEP application. Oracle CEP IDE for Eclipse offers the following features to help navigate the EPN Editor:

- [Section 6.3.1, "Moving the Canvas"](#)
- [Section 6.3.2, "Shortcuts to Component Configuration and EPN Assembly Files"](#)
- [Section 6.3.3, "Hyperlinking"](#)
- [Section 6.3.4, "Context Menus"](#)
- [Section 6.3.5, "Browsing Oracle CEP Types"](#)

6.3.1 Moving the Canvas

To move the EPN canvas without using the horizontal and vertical scroll bars, you can use any of the following options:

- Position the cursor on the canvas, hold down the middle mouse button, and drag.
- Hold down the space bar and click and drag the canvas.
- In the Overview view, click in the highlight box and drag.

6.3.2 Shortcuts to Component Configuration and EPN Assembly Files

If a node has a configuration object associated with it, then double-clicking that node will open the component configuration file where that node's behavior is defined.

Otherwise, double-clicking that node will open the EPN assembly file (the Spring context file) where that node is defined.

A configuration badge will be shown on nodes with associated configuration objects as shown in [Figure 6–20](#).

Figure 6–20 Node with Configuration Badge



For more information, see:

- [Section 6.2.7, "Configuration Badging"](#)
- [Section 6.3.3, "Hyperlinking"](#)

6.3.3 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.

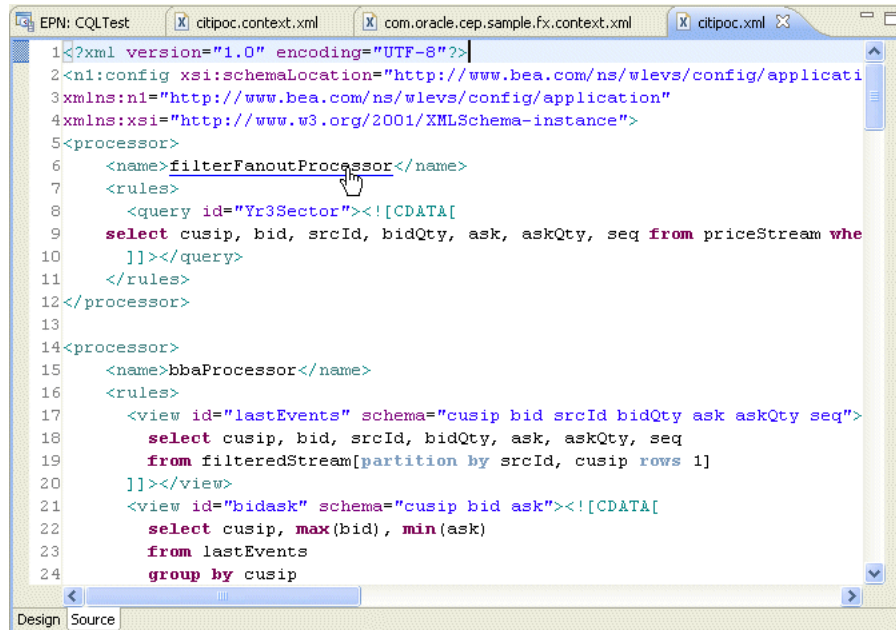
This section describes:

- [Section 6.3.3.1, "Hyperlinking in Component Configuration and EPN Assembly Files"](#)
- [Section 6.3.3.2, "Hyperlinking in Oracle CQL Statements"](#)

6.3.3.1 Hyperlinking in Component Configuration and EPN Assembly Files

Figure 6–21 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 6–22 shows.

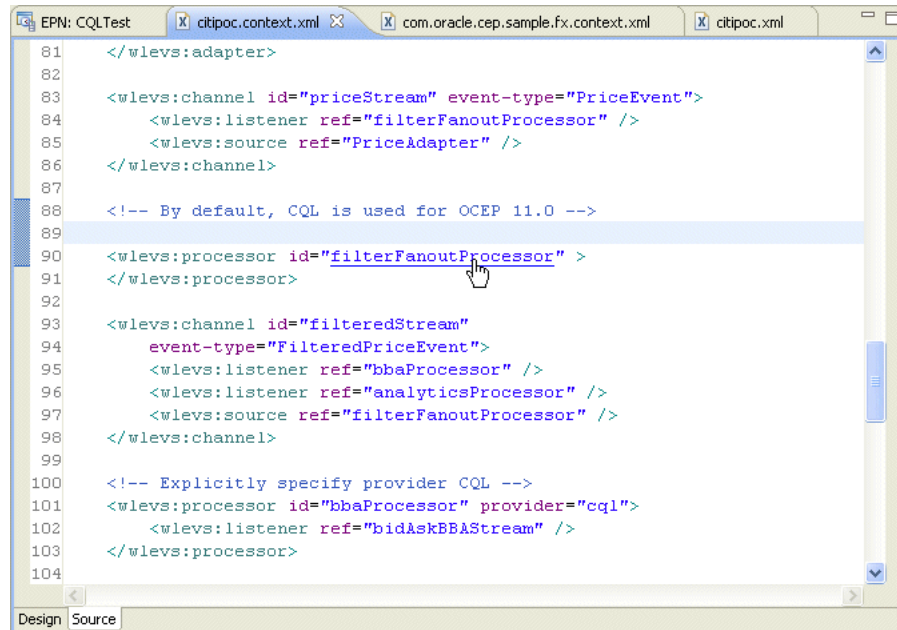
Figure 6–21 Component Configuration File: Hyperlinking to EPN Assembly File



This figure shows a component configuration file with a processor element. The mouse is hovering over the `filterFanoutProcessor` value of the name element. The name value is underlined to indicate that it is a hyperlink.

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 6–22 EPN Assembly File: Hyperlinking to Component Configuration File



6.3.3.2 Hyperlinking in Oracle CQL Statements

Figure 6–23 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 6–24 shows.

Note: Hyperlinking in Oracle SQL statements is designed for simple use cases and may not work as expected in more complex implementations.

Figure 6–23 Oracle CQL Statement: Event Schema

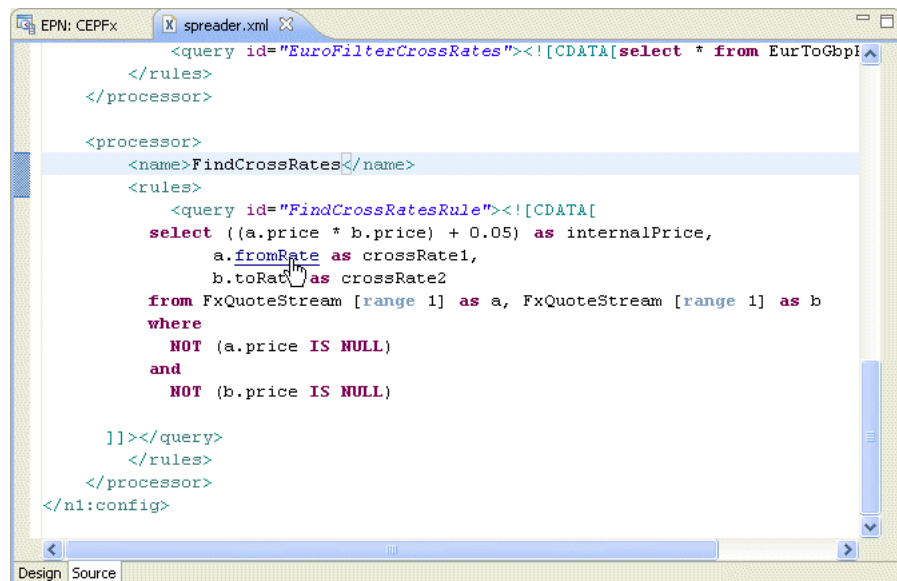
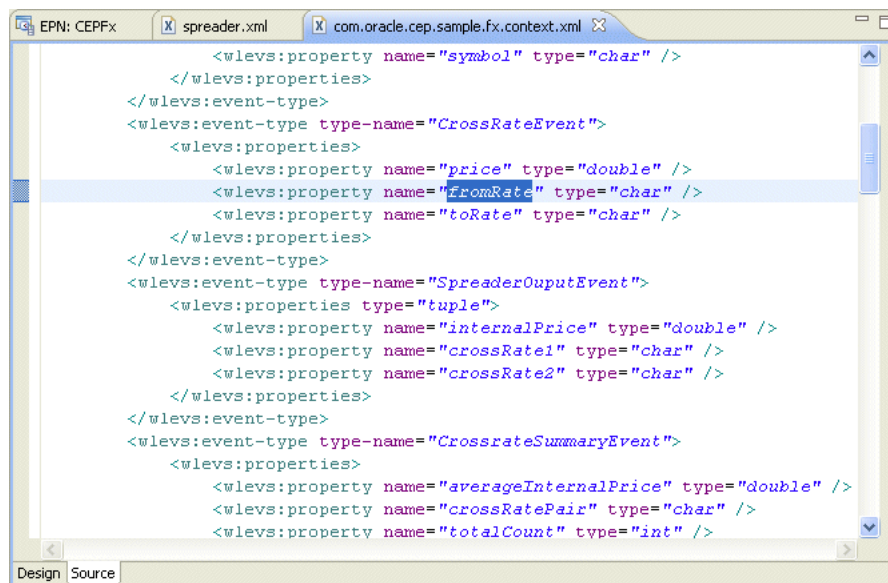


Figure 6–24 Corresponding Event Definition in EPN Assembly File


Similarly, you can **Ctrl-click** the `FxQuoteStream` channel in the Oracle CQL statement that [Figure 6–23](#) shows to jump to the channel’s definition. This is applicable wherever references to external objects are present in a Oracle CQL statement.

6.3.4 Context Menus

Each node on the EPN Editor has a group of context menu items that provide convenient access to various node-specific functions. Right-click the node to display its context menu.

Depending on the node type, you can use the EPN Editor context menu to select from the following options:

- **Go to Configuration Source:** opens the corresponding component configuration file and positions the cursor in the appropriate element. You can use hyperlinking to quickly move from this file to the corresponding EPN assembly file. For more information, see [Section 6.3.3, "Hyperlinking"](#).
- **Go to Assembly Source:** opens the corresponding EPN assembly file and positions the cursor in the appropriate element. You can use hyperlinking to quickly move from this file to the corresponding component configuration file. For more information, see [Section 6.3.3, "Hyperlinking"](#).
- **Go to Java Source:** opens the corresponding Java source file for this component.
- **Delete:** deletes the component from both the EPN assembly file and component configuration file (if applicable).
- **Rename:** allows you to change the name of the component. The name is updated in both the EPN assembly file and component configuration file (if applicable).
- **Help:** displays context sensitive help for the component.

Note that these navigation options will become disabled when a corresponding source artifact cannot be found. For example, if an adapter does not have a corresponding entry in a configuration XML file, its **Go to Configuration Source** menu item will be greyed out.

6.3.5 Browsing Oracle CEP Types

A typical Oracle CEP project contains many instances of Oracle CEP types such as adapters, channels, processors, event beans. In a large, complex Oracle CEP project, it can be a challenge to locate a particular instance. The Oracle CEP IDE for Eclipse provides an Oracle CEP type browser that you can use to quickly locate instances of any Oracle CEP type.

6.3.5.1 How to Browse Oracle CEP Types

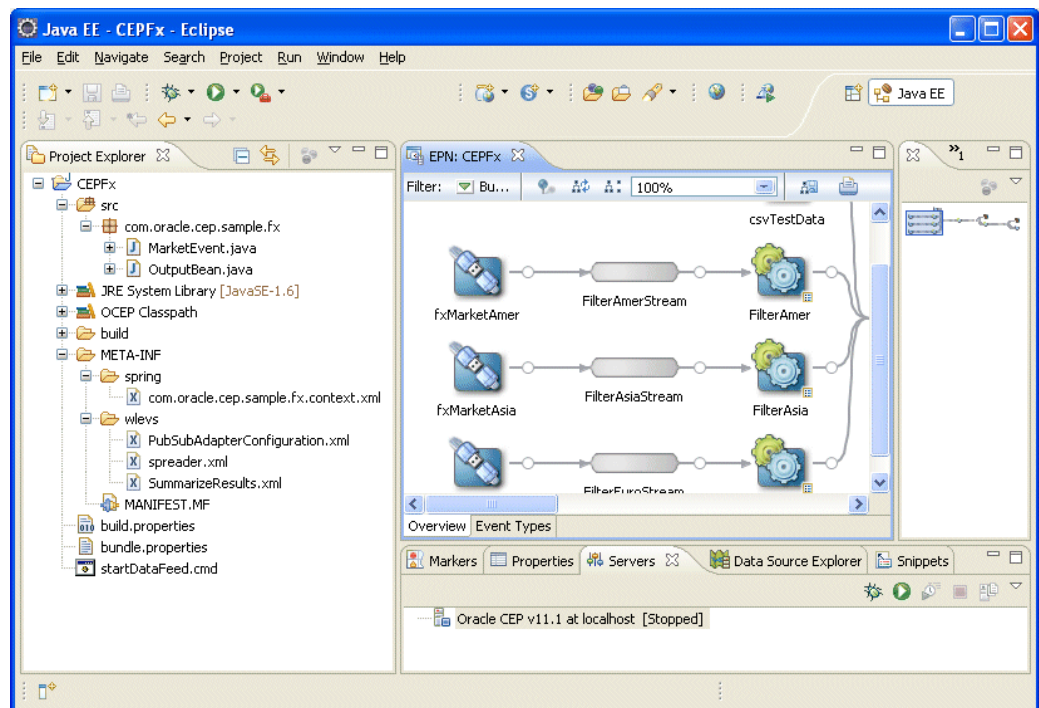
You can open the Oracle CEP type browser using the keyboard short cut **Ctrl-Alt-T**.

To browse Oracle CEP types:

1. Open an Oracle CEP project.

In the following procedure, consider the Oracle CEP project that [Figure 6–25](#) shows. This is based on the Oracle CEP foreign exchange example. For more information on this example, see "Foreign Exchange (FX) Example" in the *Oracle Complex Event Processing Getting Started*.

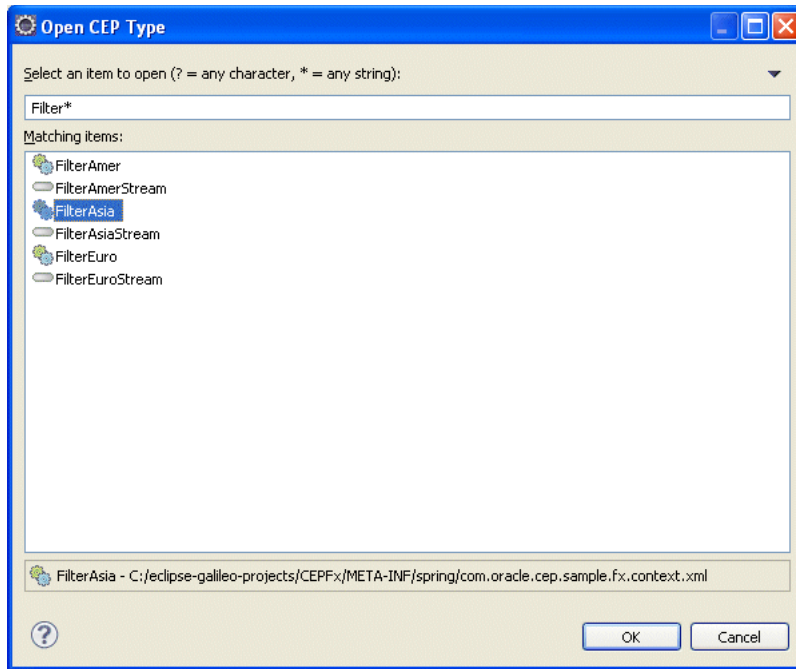
Figure 6–25 Example Oracle CEP EPN



2. Type the keyboard short cut **Ctrl-Alt-T**.

The Oracle CEP type browser appears as [Figure 6–26](#) shows.

Figure 6–26 Oracle CEP Type Browser



3. Configure the Oracle CEP Type dialog as shown in [Table 6–1](#).

Table 6–1 Oracle CEP Type Dialog

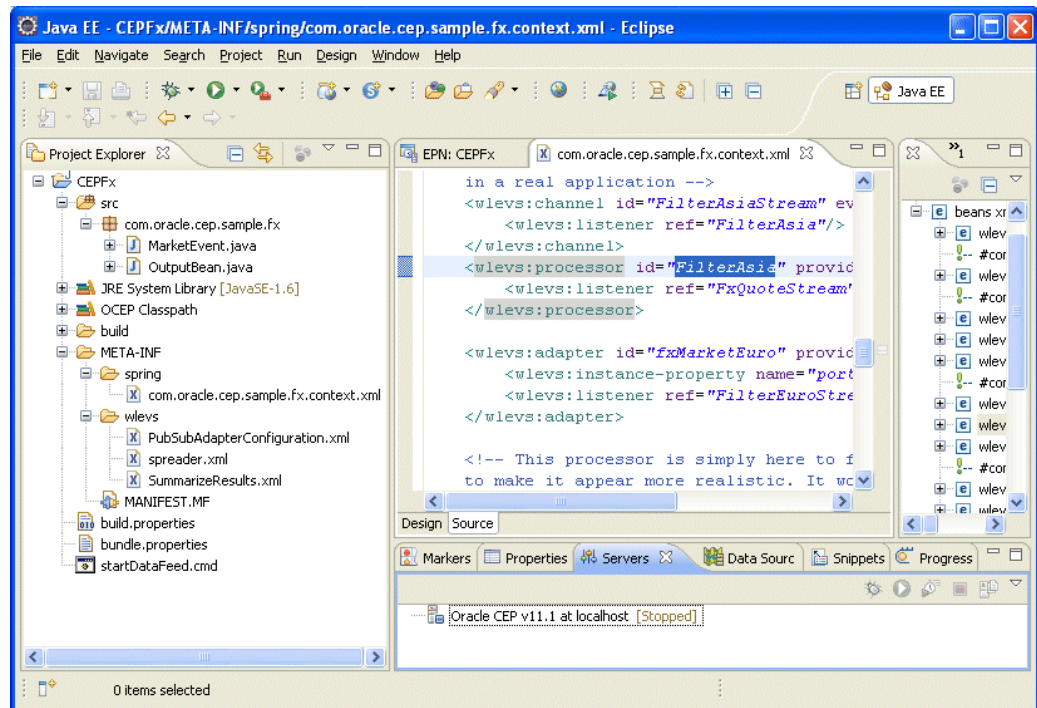
Attribute	Description
Select an item to open	Specify a filter to match the names of the items you want to find. Use the ? wildcard for any single character and the * wildcard for any string of two or more characters.
Matching items	The list of Oracle CEP type instances whose name matches the filter you 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**.

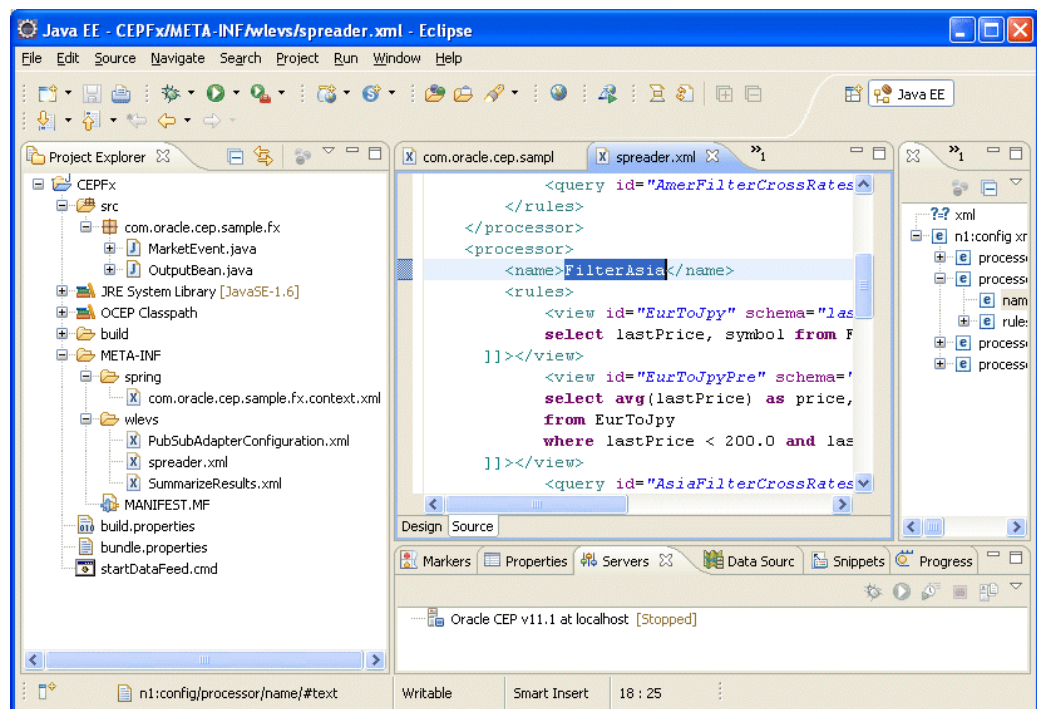
The type is opened in the source file in which it is defined. For example, selecting **FilterAsia** from the **Matching Items** list and clicking **OK** opens the `com.oracle.cep.sample.fx.content.xml` EPN assembly file in which this processor is defined as [Figure 6–27](#) shows.

Figure 6–27 Opening the FilterAsia EPN Assembly File



To navigate to the corresponding component configuration file as Figure 6–28 shows, **Ctrl-click** the **FilterAsia** id attribute value.

Figure 6–28 Opening the FilterAsia Component Configuration File



For more information on hyperlinking, see Section 6.3.3, "Hyperlinking".

6.4 Using the EPN Editor

The EPN Editor allows you to create and edit an application's EPN using actions on the editor surface. Most actions in the EPN Editor result in edits to an assembly file in that application. You can use a single EPN assembly file or multiple EPN assembly files (for more information, see [Section 6.2.2, "Filtering"](#)).

The following sections describe EPN Editor editing tasks, including:

- [Section 6.4.1, "Creating Nodes"](#)
- [Section 6.4.2, "Connecting Nodes"](#)
- [Section 6.4.3, "Laying Out Nodes"](#)
- [Section 6.4.4, "Renaming Nodes"](#)
- [Section 6.4.5, "Deleting Nodes"](#)

For more information, see:

- [Section 4.1, "Oracle CEP Project Overview"](#)
- [Section 6.2, "EPN Editor Overview"](#)
- [Section 6.1, "Opening the EPN Editor"](#)
- [Section 6.3, "Navigating the EPN Editor"](#)

6.4.1 Creating Nodes

When adding new nodes to an EPN using the EPN editor, a new node will appear at the location of the mouse click that was used to show the EPN Editor context menu. You can create any of the nodes that [Table 6–2](#) lists.

Table 6–2 EPN Editor Icons








Node	Description
	<p>Adapter: 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"> ■ Section 6.4.1.2, "How to Create an Adapter Node" ■ Chapter 7, "Configuring JMS Adapters" ■ Chapter 8, "Configuring HTTP Publish-Subscribe Server Adapters" ■ Chapter 14, "Configuring Custom Adapters" ■ Chapter 25, "Testing Applications With the Load Generator and csvgen Adapter"
	<p>Channel: 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"> ■ Section 6.4.1.1, "How to Create a Basic Node" ■ Chapter 9, "Configuring Channels"
	<p>Processor: 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"> ■ Section 6.4.1.3, "How to Create a Processor Node" ■ Chapter 10, "Configuring Oracle CQL Processors" ■ Chapter 11, "Configuring EPL Processors"

Table 6–2 (Cont.) EPN Editor Icons

Node	Description
	<p>Event Bean: a node similar to a standard Spring bean except that it can be managed by the Oracle CEP management framework and can actively use the capabilities of the Oracle CEP server container.</p> <p>For more information, see:</p> <ul style="list-style-type: none"> ▪ Section 6.4.1.1, "How to Create a Basic Node" ▪ Chapter 15, "Configuring Custom Event Beans"
	<p>Spring Bean: 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"> ▪ Section 6.4.1.1, "How to Create a Basic Node" ▪ Chapter 16, "Configuring Custom Spring Beans"
	<p>Cache: a node that provides a temporary storage area for events, created exclusively to improve the overall performance of your Oracle CEP application.</p> <p>For more information, see:</p> <ul style="list-style-type: none"> ▪ Section 6.4.1.1, "How to Create a Basic Node" ▪ Chapter 12, "Configuring Caching"
	<p>Table: 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"> ▪ Section 6.4.1.1, "How to Create a Basic Node" ▪ Section 10.3, "Configuring an Oracle CQL Processor Table Source"

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 6.4.3, "Laying Out Nodes"](#).

When a child node is nested within a parent node, its icon appears within a box. For more information, see [Section 6.2.9, "Nested Stages"](#).

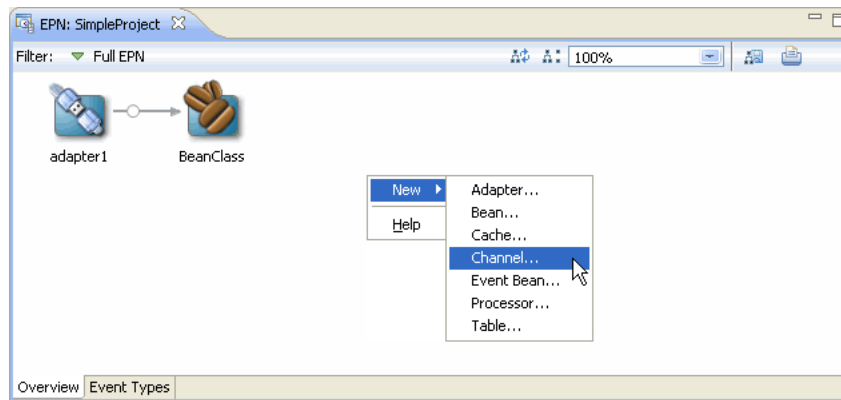
6.4.1.1 How to Create a Basic Node

Basic nodes include beans, caches, channels, event beans, and tables.

For information on how to create other nodes, see [Section 6.4.1, "Creating Nodes"](#).

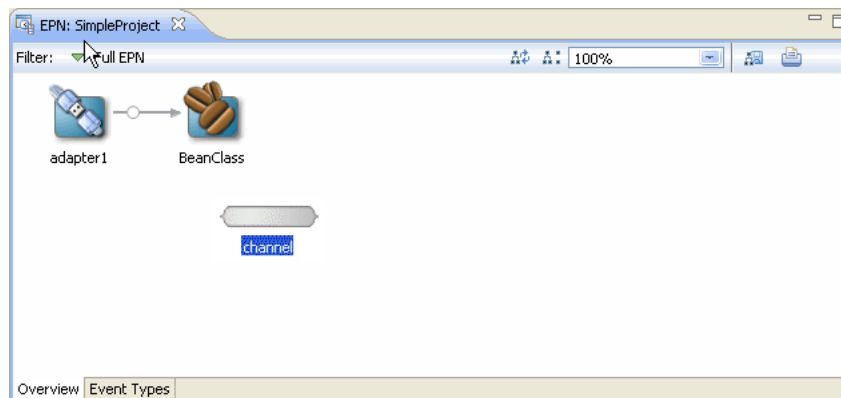
To create a basic node:

1. Open the EPN Editor (see [Section 6.1, "Opening the EPN Editor"](#)).
2. Right-click on an empty portion of the EPN Editor surface and select **New** from the context menu as [Figure 6–29](#) shows.

Figure 6–29 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 6–30](#) shows.

Figure 6–30 New Basic Node

To rename the node, see [Section 6.4.4, "Renaming Nodes"](#).

To reposition the node and update the EPN Editor layout, see [Section 6.4.3, "Laying Out Nodes"](#).

4. Optionally, configure additional node options.

See:

- [Chapter 9, "Configuring Channels"](#)
- [Section 10.3, "Configuring an Oracle CQL Processor Table Source"](#)
- [Chapter 12, "Configuring Caching"](#)
- [Chapter 14, "Configuring Custom Adapters"](#)
- [Chapter 15, "Configuring Custom Event Beans"](#)
- [Chapter 16, "Configuring Custom Spring Beans"](#)
- [Chapter 25, "Testing Applications With the Load Generator and csvgen Adapter"](#)

6.4.1.2 How to Create an Adapter Node

This section describes how to create an adapter using the EPN Editor, including:

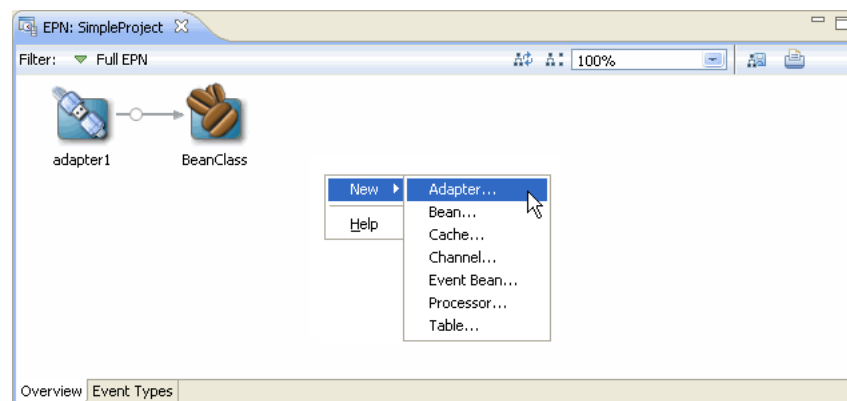
- JMS adapters (in-bound or out-bound)
- HTTP publish-subscribe server adapters (publishing or subscribing)

For information on how to create other nodes, see [Section 6.4.1, "Creating Nodes"](#).

To create an adapter node:

1. Open the EPN Editor (see [Section 6.1, "Opening the EPN Editor"](#)).
2. Right-click on an empty portion of the EPN Editor surface and select **New** from the context menu as [Figure 6–31](#) shows.

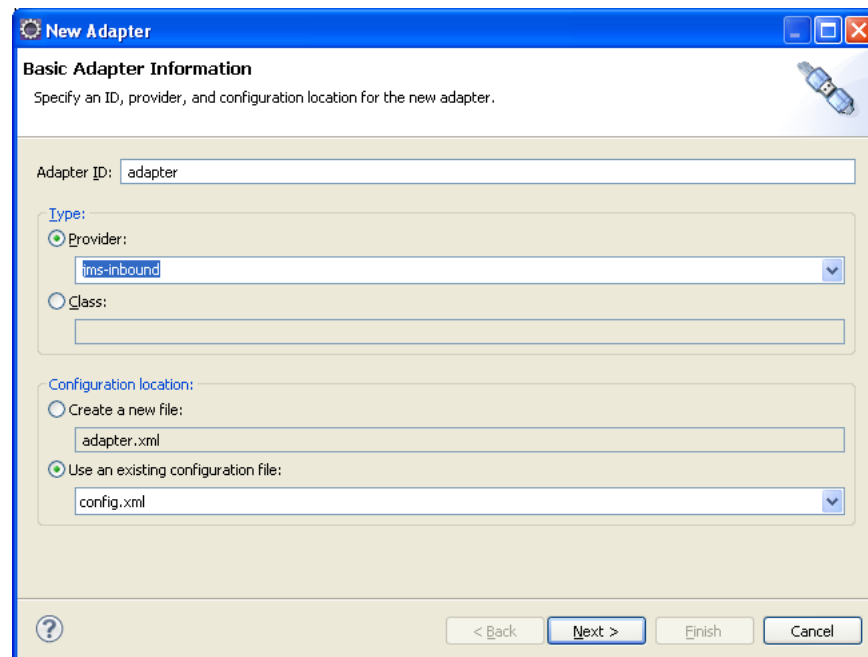
Figure 6–31 *Creating an Adapter Node*



3. Select node type **Adapter**.

The New Adapter wizard appears as shown in [Figure 6–32](#).

Figure 6–32 *New Adapter Wizard*



4. Configure the New Adapter Wizard - Page 1 as shown in [Table 6–3](#).

Table 6–3 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 CEP component configuration schema. Select one of: <ul style="list-style-type: none"> ▪ jms-inbound: JMS in-bound adapter. ▪ jms-outbound: JMS out-bound adapter. ▪ httppub: HTTP publish-suscribe adapter for publishing. ▪ httpsub: HTTP publish-suscribe adapter for subscribing.
Class	Specify the fully qualified Java class name of a custom adapter. NOTE: If you are using a custom adapter factory, you must add the <code>wlevs:factory</code> element manually. For more information, see Chapter 14, "Configuring Custom Adapters" .
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 6–33, "New Adapter Wizard - jms-inbound"](#)
See [Section 7.6.1, "JMS Inbound Adapter Component Configuration"](#).
 - [Figure 6–34, "New Adapter Wizard - jms-outbound"](#)
See [Section 7.6.2, "JMS Outbound Adapter Component Configuration"](#).
 - [Figure 6–35, "New Adapter Wizard - httppub"](#)
See [Section 8.5.1, "HTTP Pub-Sub Adapter for Publishing Component Configuration"](#).
 - [Figure 6–36, "New Adapter Wizard - httpsub"](#)

See Section 8.5.2, "HTTP Pub-Sub Adapter for Subscribing Component Configuration".

Figure 6–33 *New Adapter Wizard - jms-inbound*

New Adapter

Adapter Configuration
Enter configuration values for this adapter.

Properties:

Connection

Event Type:

JNDI Provider URL:*

JNDI Factory:

Connection JNDI Name:

Destination JNDI Name:*

Destination Name:*

Security

User:

Password:

Encrypted Password:

Connection User:

Connection Password:

Connection Encrypted Password:

Advanced

Work Manager:

Concurrent Consumers:

Message Selector:

Session Acknowledgment Mode: ▾

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

? < Back Next > Finish Cancel

Figure 6–34 New Adapter Wizard - jms-outbound

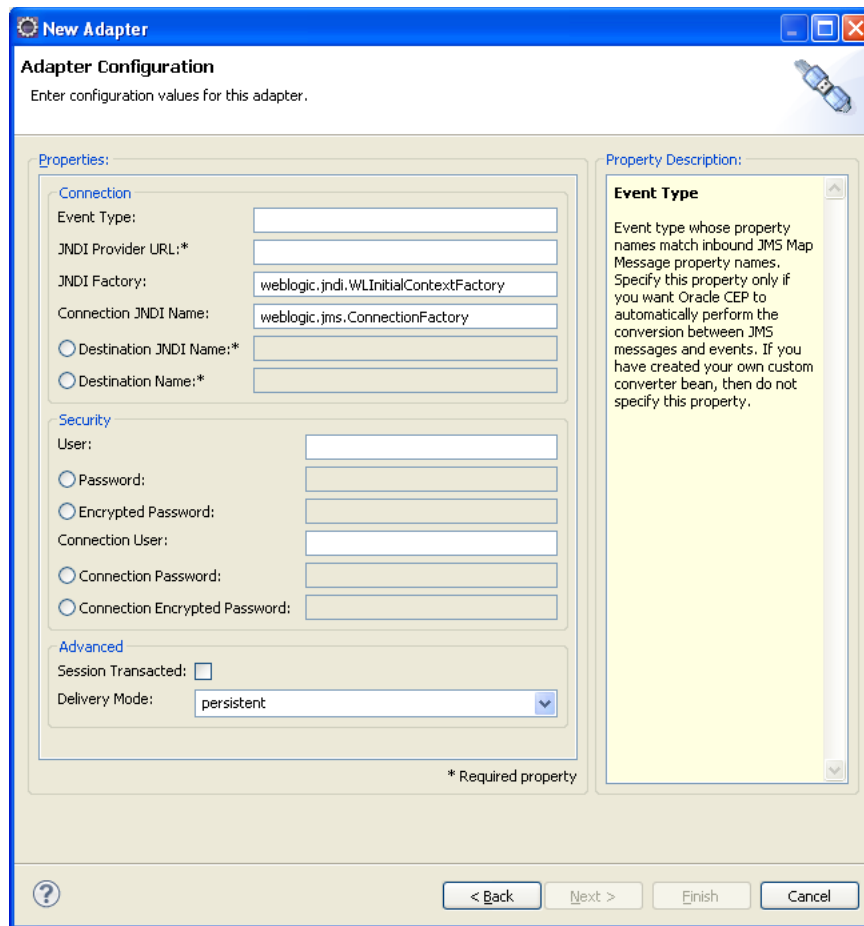


Figure 6–35 New Adapter Wizard - httppub

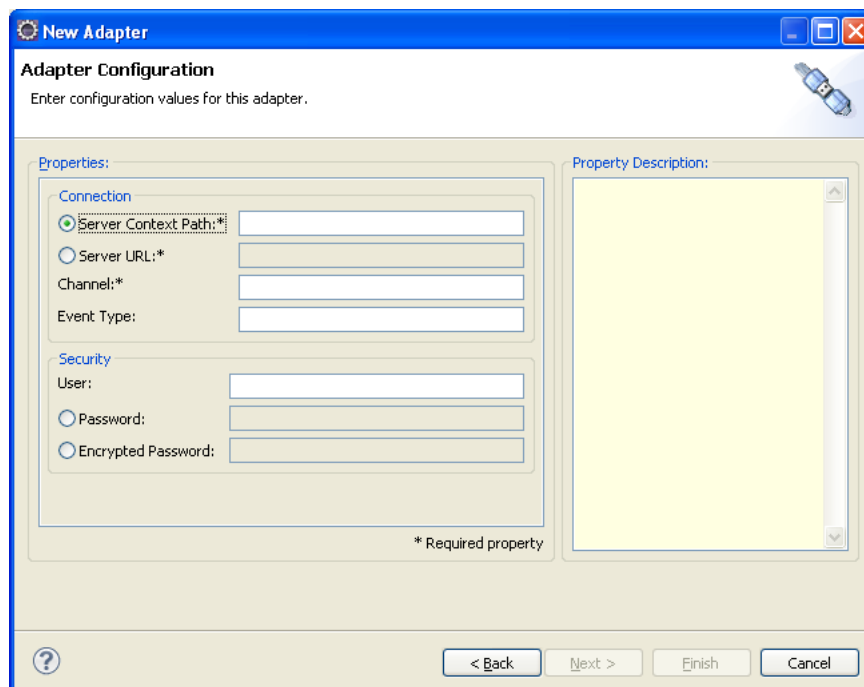


Figure 6–36 New Adapter Wizard - https

New Adapter

Adapter Configuration
Enter configuration values for this adapter.

Properties:

Connection

Server URL:*

Channel:*

Event Type:

Security

User:

Password:

Encrypted Password:

* Required property

Property Description:

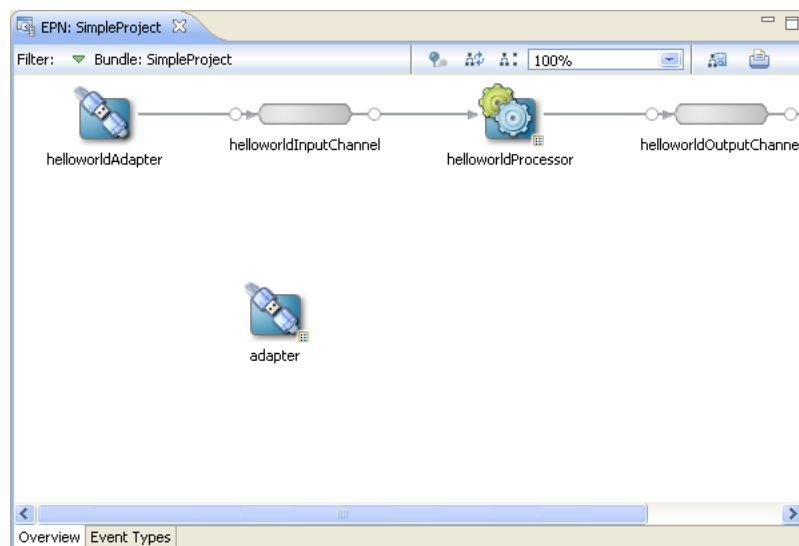
Server URL

The Server URL is used when publishing or subscribing to a remote channel, i.e. a channel hosted by a remote Pub-Sub server. The Server URL identifies the remote Pub-Sub server. For example, a value such as 'http://myhost.com:9002/pubsub' could be used to identify the Pub-Sub server running on host myhost.com at port 9002.

? < Back Next > Finish Cancel

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 6–37](#) shows.

Figure 6–37 New Adapter Node

To rename the node, see [Section 6.4.4, "Renaming Nodes"](#).

To reposition the node and update the EPN Editor layout, see [Section 6.4.3, "Laying Out Nodes"](#).

10. Optionally, configure additional node options.

For more information, see:

- [Chapter 7, "Configuring JMS Adapters"](#)
- [Chapter 8, "Configuring HTTP Publish-Subscribe Server Adapters"](#)

6.4.1.3 How to Create a Processor Node

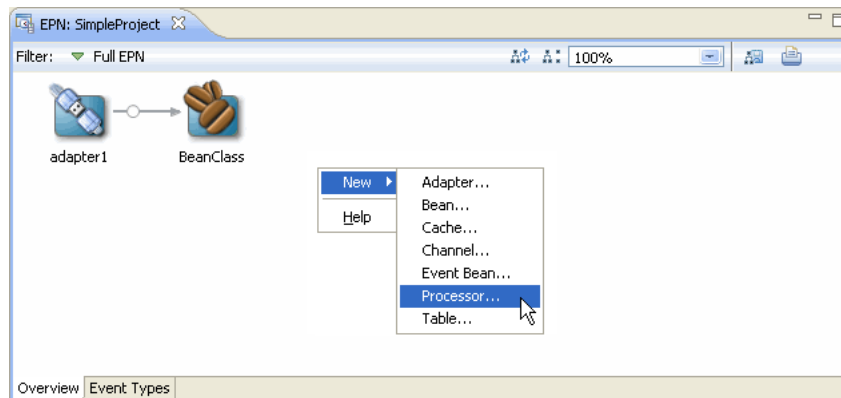
This section describes how to create a processor node using the EPN Editor. For information on creating other node types, see [Section 6.4.1.1, "How to Create a Basic Node"](#).

When deploying an Oracle CEP application with a `wlevs:processor` node, other nodes in an EPN may reference that processor only if a processor configuration exists for that processor. Processor configurations are defined in Oracle CEP application configuration files. See [Section 1.1.5, "Component Configuration Files"](#) for more information about CEP configuration files.

To create a processor node:

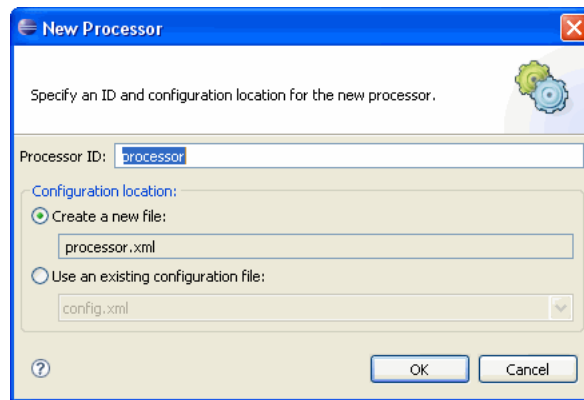
1. Open the EPN Editor (see [Section 6.1, "Opening the EPN Editor"](#)).
2. Right-click on an empty portion of the EPN Editor surface and select **New** from the context menu as [Figure 6–38](#) shows.

Figure 6–38 *Creating a Processor Node*



3. Select node type **Processor**.

The New Processor dialog appears as shown in [Figure 6–39](#).

Figure 6–39 New Processor Dialog

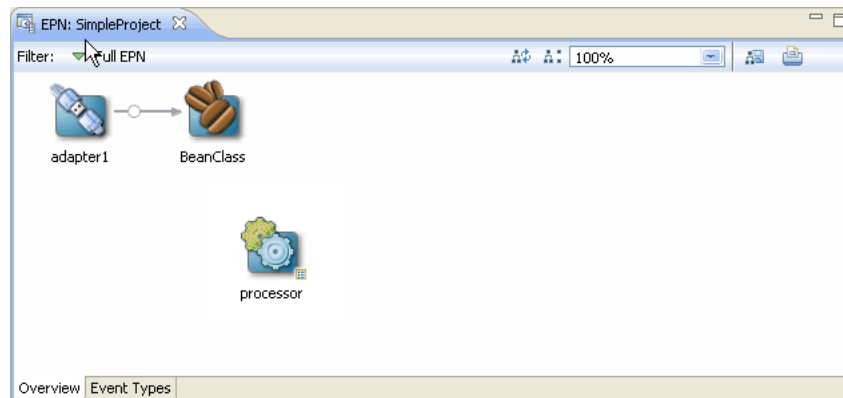
4. Configure the New Processor dialog as shown in [Table 6–4](#).

Table 6–4 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/wl evs 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 6–40](#) shows.

Figure 6–40 New Processor Node

To rename the node, see [Section 6.4.4, "Renaming Nodes"](#).

To reposition the node and update the EPN Editor layout, see [Section 6.4.3, "Laying Out Nodes"](#).

Note: In Oracle CEP, 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 9.1.2, "Channels Representing Streams and Relations"](#).

6. Optionally, configure additional processor options.

See:

- [Chapter 10, "Configuring Oracle CQL Processors"](#)
- [Chapter 11, "Configuring EPL Processors"](#)

6.4.2 Connecting Nodes

The nodes in the EPN represent the flow of events through an Event Processing Network of an Oracle CEP application. When a node may forward events to another node in the EPN, the EPN Editor allows you to connect that node visually by dragging a line from the source node to the destination node.

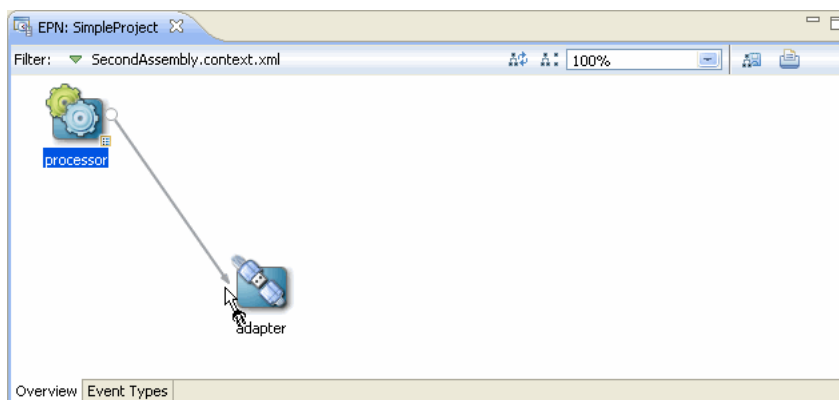
6.4.2.1 How to Connect Nodes

This section describes how to connect nodes in the EPN Editor.

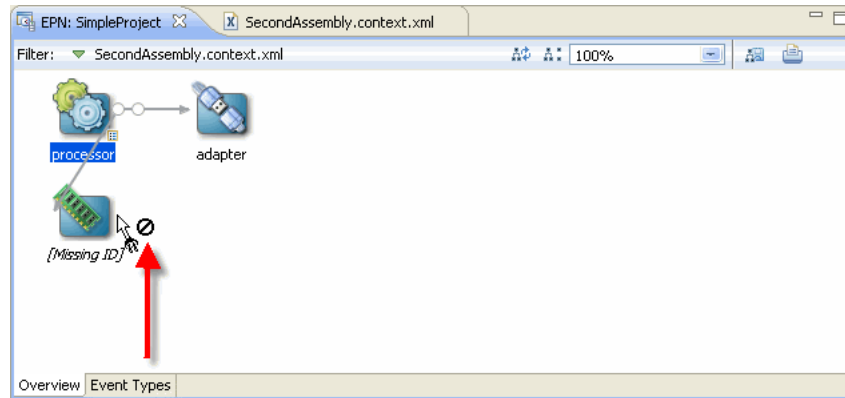
To connect nodes:

1. Open the EPN Editor (see [Section 6.1, "Opening the EPN Editor"](#)).
2. Select the source of events and drag to the target of the event flow.
 - If a connection is allowed, a plug icon is shown at the target end as [Figure 6–41](#) shows.

Figure 6–41 Connecting Nodes: Connection Allowed



- If the connection is not allowed, a forbidden icon is shown at the target end as [Figure 6–42](#) shows.

Figure 6–42 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 6.2.9, "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 6–43](#).

Figure 6–43 Valid Connections

[Figure 6–44](#) shows the EPN assembly file before connection.

Figure 6–44 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 6–45](#) shows the EPN assembly file after connection.

Figure 6–45 EPN Assembly File: After Connection

```

<wlevs:adapter id="adapter" provider="httppub">
  <wlevs:listener ref="channel" />
</wlevs:adapter>

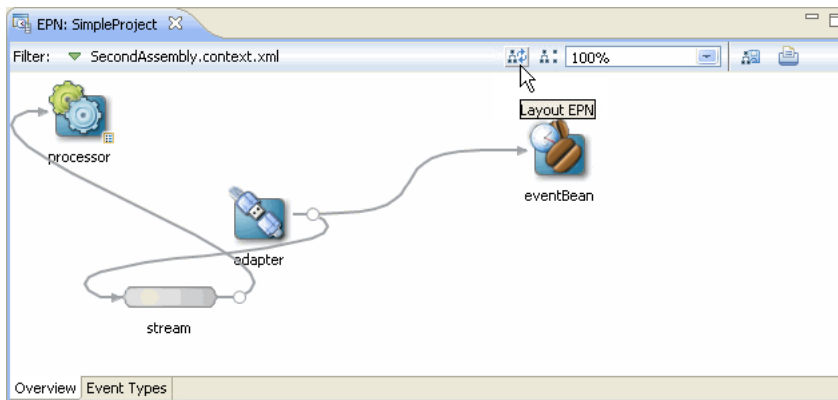
<wlevs:channel id="channel">
  <wlevs:listener ref="processor" />
</wlevs:channel>

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

```

6.4.3 Laying Out Nodes

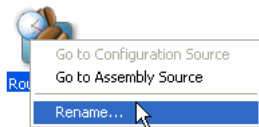
Certain EPN Editor actions will use the location of the action as the location of the rendered result. For example, when adding new nodes to an EPN using the EPN editor, a new node will appear at the location of the mouse click that was used to show the EPN Editor context menu. The user may not reposition the nodes on the EPN Editor. To refresh the layout of the nodes on the EPN Editor, click the **Layout EPN** button on the EPN Editor toolbar as [Figure 6–46](#) shows.

Figure 6–46 Laying Out Nodes

For more information, see [Section 6.2.4, "Layout"](#).

6.4.4 Renaming Nodes

Most node types support a rename operation that will change all references to the node across both assembly and configuration XML files. You can select **Rename** from the node's context menu as [Figure 6–47](#) shows.

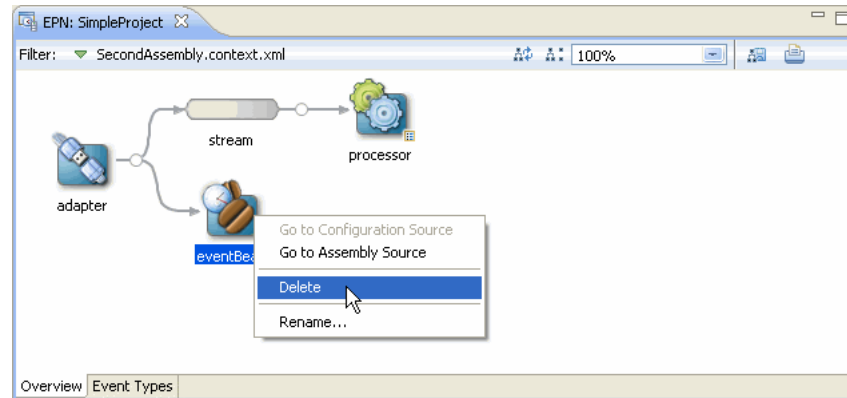
Figure 6–47 Renaming Nodes

6.4.5 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 6-48](#) shows.

Figure 6-48 Deleting Nodes



When deleting a node, the incoming and outgoing connections are also deleted. For example, [Figure 6-49](#) shows the EPN and [Figure 6-51](#) shows the assembly file before deleting the channel node named `stream`.

Figure 6-49 EPN Before Deleting a Channel Node

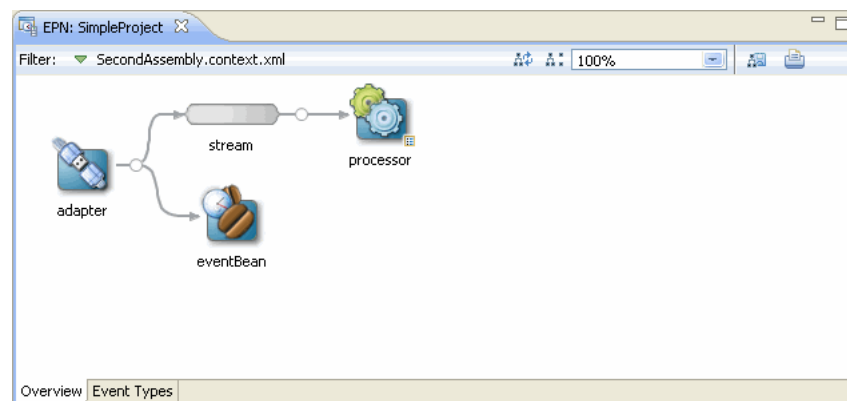


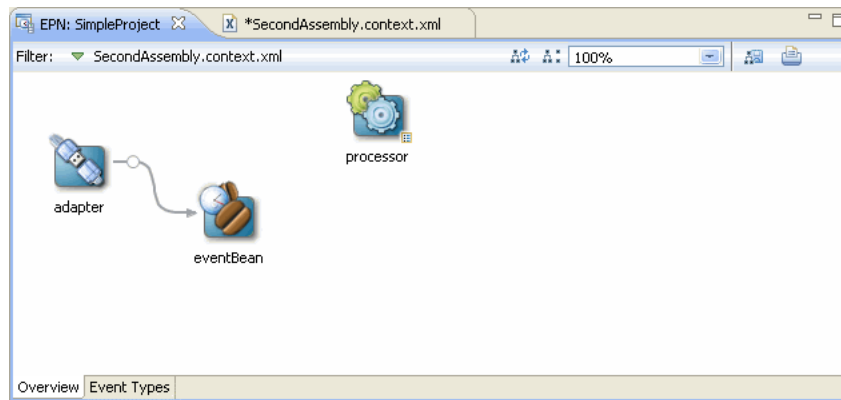
Figure 6-50 Assembly File Before Deleting a Channel Node

```
<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 6-51](#) shows the EPN and [Figure 6-52](#) shows the assembly file after deleting this channel node.

Figure 6–51 EPN After Deleting a Channel Node**Figure 6–52 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

Building the Oracle CEP Event Processing Network

Part III contains the following chapters:

- Chapter 7, "Configuring JMS Adapters"
- Chapter 8, "Configuring HTTP Publish-Subscribe Server Adapters"
- Chapter 9, "Configuring Channels"
- Chapter 10, "Configuring Oracle CQL Processors"
- Chapter 11, "Configuring EPL Processors"
- Chapter 12, "Configuring Caching"
- Chapter 13, "Configuring Event Record and Playback"

Configuring JMS Adapters

This section contains information on the following subjects:

- [Section 7.1, "Overview of JMS Adapter Configuration"](#)
- [Section 7.2, "Configuring a JMS Adapter for a JMS Service Provider"](#)
- [Section 7.3, "Creating a Custom Converter Between JMS Messages and Event Types"](#)
- [Section 7.4, "Encrypting Passwords in the JMS Adapter Component Configuration File"](#)
- [Section 7.5, "Configuring the JMS Adapter EPN Assembly File"](#)
- [Section 7.6, "Configuring the JMS Adapter Component Configuration File"](#)

7.1 Overview of JMS Adapter Configuration

The Oracle CEP JMS adapters support any Java EE compliant JMS service provider that provides a Java client. For more information, see [Section 7.1.1, "JMS Service Providers"](#).

Oracle CEP 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 7.1.2, "Inbound JMS Adapter"](#)
- [Section 7.1.3, "Outbound JMS Adapter"](#)

For general information about JMS, see Java Message Service on the Sun Developer Network at <http://java.sun.com/products/jms/>.

7.1.1 JMS Service Providers

The Oracle CEP JMS adapters support any Java EE compliant JMS service provider that provides a Java client.

This chapter describes how to configure the Oracle CEP JMS inbound and outbound adapters for use with the following JMS service providers:

- [Section 7.2.3, "How to Configure a JMS Adapter for Oracle WebLogic Server JMS Manually"](#)
- [Section 7.2.4, "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 CEP JMS adapters for use with your JMS service provider. Review the procedure in

[Section 7.2.4, "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 7.2, "Configuring a JMS Adapter for a JMS Service Provider"](#).

7.1.2 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 7.1.2.1, "Conversion Between JMS Messages and Event Types"](#)
- [Section 7.1.2.2, "Single and Multi-threaded Inbound JMS Adapters"](#)

For more information, see:

- [Section 7.5.1, "JMS Inbound Adapter EPN Assembly File Configuration"](#)
- [Section 7.6.1, "JMS Inbound Adapter Component Configuration"](#)

7.1.2.1 Conversion Between JMS Messages and Event Types

By default, the inbound JMS adapter can automatically convert 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 CEP 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 7.3, "Creating a Custom Converter Between JMS Messages and Event Types"](#).

7.1.2.2 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 7-1, "jms-adapter Inbound Child Elements"](#)
- [Section 22.2.1, "EventPartitioner"](#)

7.1.3 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 CEP 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 CEP 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 7.5.2, "JMS Outbound Adapter EPN Assembly File Configuration"](#)
- [Section 7.6.2, "JMS Outbound Adapter Component Configuration"](#)
- [Section 7.3, "Creating a Custom Converter Between JMS Messages and Event Types."](#)

7.2 Configuring a JMS Adapter for a JMS Service Provider

This section describes how to configure the Oracle CEP JMS inbound and outbound adapters:

- [Section 7.2.1, "How to Configure a JMS Adapter Using the Oracle CEP IDE for Eclipse"](#)
- [Section 7.2.2, "How to Configure a JMS Adapter Manually"](#)

This section provides examples specific to the following JMS service providers:

- [Section 7.2.3, "How to Configure a JMS Adapter for Oracle WebLogic Server JMS Manually"](#)
- [Section 7.2.4, "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 CEP JMS adapters for use with your JMS service provider. Review the procedure in [Section 7.2.4, "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 7.1.1, "JMS Service Providers"](#).

Note: In the following sections, it is assumed that you have already created an Oracle CEP application along with its EPN assembly file and component configuration files, and that you want to update the application to use an inbound or outbound JMS adapter. If you have not, refer to [Chapter 1, "Overview of Creating Oracle CEP Applications"](#) for details.

7.2.1 How to Configure a JMS Adapter Using the Oracle CEP IDE for Eclipse

The simplest way to create and configure a JMS adapter is using the Oracle CEP IDE for Eclipse adapter wizard.

For more information, see [Section 6.4.1.2, "How to Create an Adapter Node"](#).

After using the adapter wizard to create and specify the basic JMS adapter configuration, review [Section 7.2.2, "How to Configure a JMS Adapter Manually"](#) to complete the configuration.

7.2.2 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 CEP IDE for Eclipse adapter wizard as [Section 7.2.1, "How to Configure a JMS Adapter Using the Oracle CEP 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 7-1](#) shows the `wlevs:adapter` element for a JMS inbound adapter.

Example 7-1 *wlevs:adapter Element for Inbound Adapter*

```
<wlevs:adapter id="inboundJmsAdapter1" provider="jms-inbound">
...
</wlevs:adapter>
```

See:

- [Section 7.5.1, "JMS Inbound Adapter EPN Assembly File Configuration"](#)
 - [Section 7.5.2, "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 7-2](#) shows the `jms-adapter` element for the JMS inbound adapter in [Example 7-1](#).

Example 7-2 *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 7.6.1, "JMS Inbound Adapter Component Configuration"](#)
 - [Section 7.6.2, "JMS Outbound Adapter Component Configuration"](#)
3. Decide how you want to convert between JMS messages and Oracle CEP 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 7.6.1, "JMS Inbound Adapter Component Configuration"](#)
- [Section 7.6.2, "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 7.3, "Creating a Custom Converter Between JMS Messages and Event Types"](#).

- 4. Configure the `jms-adapter` elements for your JMS provider as [Example 7-3](#) shows:

Example 7-3 `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 CEP JMS adapters support, see:

- [Section 7.6.1, "JMS Inbound Adapter Component Configuration"](#)
- [Section 7.6.2, "JMS Outbound Adapter Component Configuration"](#)

For specific JMS provider examples, see:

- [Section 7.2.3, "How to Configure a JMS Adapter for Oracle WebLogic Server JMS Manually"](#)
- [Section 7.2.4, "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 7.4, "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 24.1.3.3, "Creating Application Libraries"](#).

For a specific JMS provider example, see [Section 7.2.4, "How to Configure a JMS Adapter for Tibco EMS JMS Manually"](#).

7. Copy the JMS client JAR application library to the appropriate Oracle CEP server application library directory:
 - a. If your bundle is a driver, you must put it in the library extensions directory. See [Section 24.1.3.2, "Library Extensions Directory"](#).
 - b. If your bundle is not a driver, you may put it in the library directory. See [Section 24.1.3.1, "Library Directory"](#)

For more information, see [Section 24.3.4, "How to Update an Application Library Using Oracle CEP 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 4.7.5, "How to Import a Package"](#).

7.2.3 How to Configure a JMS Adapter for Oracle WebLogic Server JMS Manually

Oracle CEP includes a WebLogic JMS client.

When connecting to Oracle WebLogic server, Oracle CEP uses the T3 client by default.

You can use the IIOP WebLogic client by starting Oracle WebLogic Server with the `-useIIOP` command-line argument. This is a server-wide setting that is independent of the JMS code being used (whether it is one of the provided adapters or custom JMS code).

It is not possible to mix T3 and IIOP usage within a running Oracle CEP server.

For more information, see [Section 7.2, "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 CEP IDE for Eclipse adapter wizard as [Section 7.2.1, "How to Configure a JMS Adapter Using the Oracle CEP 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 7-4](#) shows the `wlevs:adapter` element for a JMS inbound adapter.

Example 7-4 *wlevs:adapter Element for Inbound Adapter*

```
<wlevs:adapter id="inboundJmsAdapter1" provider="jms-inbound">
...
</wlevs:adapter>
```

See:

- [Section 7.5.1, "JMS Inbound Adapter EPN Assembly File Configuration"](#)
 - [Section 7.5.2, "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 7-5](#) shows the `jms-adapter` element for the JMS inbound adapter in [Example 7-4](#).

Example 7-5 `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 7.6.1, "JMS Inbound Adapter Component Configuration"](#)
 - [Section 7.6.2, "JMS Outbound Adapter Component Configuration"](#)
3. Decide how you want to convert between JMS messages and Oracle CEP 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 7.6.1, "JMS Inbound Adapter Component Configuration"](#)
- [Section 7.6.2, "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 7.3, "Creating a Custom Converter Between JMS Messages and Event Types"](#).

4. Configure the `jms-adapter` elements for your Oracle WebLogic Server JMS provider.

[Example 7-6](#) shows the `jms-adapter` elements for a JMS inbound and JMS outbound adapter.

Example 7-6 `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 7.6.1, "JMS Inbound Adapter Component Configuration"](#)
 - [Section 7.6.2, "JMS Outbound Adapter Component Configuration"](#)
5. If you specify JMS provider client passwords in the component configuration file, consider encrypting them.

See [Section 7.4, "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 24.1.3.3, "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 CEP server application library directory:
- a. If your bundle is a driver, you must put it in the library extensions directory. See [Section 24.1.3.2, "Library Extensions Directory"](#).
 - b. If your bundle is not a driver, you may put it in the library directory. See [Section 24.1.3.1, "Library Directory"](#)

For more information, see [Section 24.3.4, "How to Update an Application Library Using Oracle CEP 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 4.7.5, "How to Import a Package"](#).

7.2.4 How to Configure a JMS Adapter for Tibco EMS JMS Manually

Oracle CEP 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 CEP server library directory:

- `tibjms.jar`

For more information, see:

- [Section 7.2, "Configuring a JMS Adapter for a JMS Service Provider"](#)
- [Section 24.1.3.1, "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 CEP IDE for Eclipse adapter wizard as [Section 7.2.1, "How to Configure a JMS Adapter Using the Oracle CEP 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 7-7](#) shows the `wlevs:adapter` element for a JMS inbound adapter.

Example 7-7 *wlevs:adapter Element for Inbound Adapter*

```
<wlevs:adapter id="inboundJmsAdapter1" provider="jms-inbound">
...
</wlevs:adapter>
```

See:

- [Section 7.5.1, "JMS Inbound Adapter EPN Assembly File Configuration"](#)
 - [Section 7.5.2, "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 7-8](#) shows the `jms-adapter` element for the JMS inbound adapter in [Example 7-7](#).

Example 7-8 *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 7.6.1, "JMS Inbound Adapter Component Configuration"](#)

- [Section 7.6.2, "JMS Outbound Adapter Component Configuration"](#)
- 3. Decide how you want to convert between JMS messages and Oracle CEP 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 7.6.1, "JMS Inbound Adapter Component Configuration"](#)
 - [Section 7.6.2, "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 7.3, "Creating a Custom Converter Between JMS Messages and Event Types"](#).
- 4. Configure the `jms-adapter` elements for your Tibco EMS JMS provider as [Example 7-9](#) shows:

Example 7-9 `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 7.6.1, "JMS Inbound Adapter Component Configuration"](#)
- [Section 7.6.2, "JMS Outbound Adapter Component Configuration"](#)
- 5. If you specify JMS provider client passwords in the component configuration file, consider encrypting them.

See [Section 7.4, "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 24.1.3.3, "Creating Application Libraries"](#).

For a specific JMS provider example, see [Section 7.2.4, "How to Configure a JMS Adapter for Tibco EMS JMS Manually"](#).

7. Copy the application library to the appropriate Oracle CEP server application library directory:
 - a. If your bundle is a driver, you must put it in the library extensions directory. See [Section 24.1.3.2, "Library Extensions Directory"](#).
 - b. If your bundle is not a driver, you may put it in the library directory. See [Section 24.1.3.1, "Library Directory"](#)

For more information, see [Section 24.3.4, "How to Update an Application Library Using Oracle CEP 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 4.7.5, "How to Import a Package"](#).

7.3 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 7.3.1, "How to Create a Custom Converter for the Inbound JMS Adapter"](#)
- [Section 7.3.2, "How to Create a Custom Converter for the Outbound JMS Adapter"](#)

7.3.1 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, JMSException;
```

The message parameter corresponds to the incoming JMS message and the return value is a List of events that will be passed on to the next stage of the event processing network.

See the *Oracle Fusion Middleware Java API Reference for Oracle Complex Event Processing* for a full description of these APIs.

To create a custom converter for the inbound JMS adapter:

1. Using the Oracle CEP 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 7–10](#) shows a possible implementation.

Example 7–10 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 7–11](#) 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 7–11 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 24, "Assembling and Deploying Oracle CEP Applications"](#).

7.3.2 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 Complex Event Processing* for a full description of these APIs.

To create a custom converter for the outbound JMS adapter:

1. Using the Oracle CEP 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 7–10](#) shows a possible implementation.

Example 7–12 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.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 7-11](#) 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 7-13 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 24, "Assembling and Deploying Oracle CEP Applications"](#).

7.4 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.

7.4.1 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 "Setting Your Development Environment" in the *Oracle Complex Event Processing Getting Started*.
2. Change to the directory that contains the configuration file for your JMS adapter.
3. Execute the following `encryptMSAConfig` command to encrypt the value of the `<password>` element in the configuration file:

```

prompt> ORACLE_CEP_HOME/ocep_11.1/bin/encryptMSAConfig . config_file
msainternal.dat_file

```

where `ORACLE_CEP_HOME` refers to the main BEA directory into which you installed Oracle CEP, such as `d:\oracle_cep`. The second argument refers to the

directory that contains the JMS adapter configuration file; because this procedure directs you to actually change to the directory, the example shows ". ". The *config_file* parameter refers to the name of your JMS adapter configuration file. Finally, the *msainternal.dat_file* parameter refers to the location of the *.msainternal.dat* file associated with your domain; by default this file is located in the *DOMAIN_DIR/servername* directory, where *DOMAIN_DIR* refers to the domain directory such as */oracle_cep/user_projects/domains/mydomain* and *servername* refers to the server instance.

The `encryptMSAConfig` command comes in two flavors: `encryptMSAConfig.cmd` (Windows) and `encryptMSAConfig.sh` (UNIX).

After you run the command, the value of the `<password>` element will be encrypted, as shown in bold in the following example:

```
<jms-adapter>
  <name>jmsInbound</name>
  <jndi-provider-url>t3://localhost:7001</jndi-provider-url>
  <destination-jndi-name>Queue1</destination-jndi-name>
  <user>weblogic</user>
  <password>{Salted-3DES}B7L6nehu7dgPtJJTnTJWRA==</password>
  <work-manager>JettyWorkManager</work-manager>
  <concurrent-consumers>1</concurrent-consumers>
  <session-transacted>>false</session-transacted>
</jms-adapter>
```

- Using your favorite XML editor, edit the JMS adapter configuration file. Change the `<password>` element (whose value is now encrypted) to `<encrypted-password>`, as shown in bold in the following example:

```
<jms-adapter>
  <name>jmsInbound</name>
  <jndi-provider-url>t3://localhost:7001</jndi-provider-url>
  <destination-jndi-name>Queue1</destination-jndi-name>
  <user>weblogic</user>
  <encrypted-password>{Salted-3DES}B7L6nehu7dgPtJJTnTJWRA==</encrypted-password>
  >
  <work-manager>JettyWorkManager</work-manager>
  <concurrent-consumers>1</concurrent-consumers>
  <session-transacted>>false</session-transacted>
</jms-adapter>
```

7.5 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 CEP 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 7.5.1, "JMS Inbound Adapter EPN Assembly File Configuration"](#)
- [Section 7.5.2, "JMS Outbound Adapter EPN Assembly File Configuration"](#)

For more information, see:

- [Section 1.1.5, "Component Configuration Files."](#)
- [Section B.2, "Component Configuration Schema wlevs_application_config.xsd"](#)

7.5.1 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 CEP 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_3.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>

```

7.5.2 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 CEP 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 CEP 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_3.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>

```


7.6 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 `wlavs:adapter` element in the EPN assembly file that declares this adapter.

This section describes:

- [Section 7.6.1, "JMS Inbound Adapter Component Configuration"](#)
- [Section 7.6.2, "JMS Outbound Adapter Component Configuration"](#)

For more information, see:

- [Section 1.1.5, "Component Configuration Files."](#)
- [Section B.2, "Component Configuration Schema `wlavs_application_config.xsd`"](#)

7.6.1 JMS Inbound Adapter Component Configuration

[Table 7–1](#) lists the `jms-adapter` element child elements applicable to the JMS inbound adapter.

Table 7–1 *jms-adapter Inbound Child Elements*

Child Element	Description
<code>bindings</code>	<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 CEP 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 Section 22.2.2, "ActiveActiveGroupBean"</p>
<code>concurrent-consumers</code>	<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"> ▪ Consider the work-manager configuration. ▪ Be sure that your converter bean is thread-safe because the converter bean will be shared among the consumers. For more information, see Section 7.3, "Creating a Custom Converter Between JMS Messages and Event Types".
<code>connection-jndi-name</code>	<p>Optional. The JNDI name of the JMS connection factory. Default value is <code>weblogic.jms.ConnectionFactory</code>, for Oracle CEP server JMS.</p>
<code>connection-password</code> <code>connection-encrypted-password</code>	<p>Optional. Either the password, or encrypted password, for <code>connection-user</code>.</p> <p>Note: Specify either <code>connection-password</code> or <code>connection-encrypted-password</code>, but not both.</p> <p>See Section 7.4, "Encrypting Passwords in the JMS Adapter Component Configuration File" for details on encrypting the password.</p>

Table 7-1 (Cont.) jms-adapter Inbound Child Elements

Child Element	Description
connection-user	<p>Optional. When Oracle CEP calls the <code>createConnection</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 CEP uses the <code>user</code> and <code>password</code> (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>Note: 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>Note: Specify either <code>destination-jndi-name</code> or <code>destination-name</code>, but not both.</p>
event-type	<p>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</p> <p>For more information, see Section 7.3, "Creating a Custom Converter Between JMS Messages and Event Types".</p>
jndi-factory	<p>Optional. The JNDI factory name. Default value is <code>weblogic.jndi.WLInitialContextFactory</code>, for Oracle CEP 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>Note: Specify either <code>password</code> or <code>encrypted-password</code>, but not both.</p> <p>See Section 7.4, "Encrypting Passwords in the JMS Adapter Component Configuration File" 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"> ▪ <code>AUTO_ACKNOWLEDGE</code>: 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. ▪ <code>CLIENT_ACKNOWLEDG</code>: With this acknowledgment mode, the client acknowledges a consumed message by calling the message's <code>acknowledge</code> method. ▪ <code>DUPS_OK_ACKNOWLEDGE</code>: This acknowledgment mode instructs the session to lazily acknowledge the delivery of messages. <p>Default: <code>AUTO_ACKNOWLEDGE</code>.</p>

Table 7-1 (Cont.) jms-adapter Inbound Child Elements

Child Element	Description
session-transacted	Boolean value that specifies whether or not the session is transactional. If the session is transacted then do not specify <code>session-ack-mode-name</code> . Default: <code>False</code> .
user	Required. When Oracle CEP acquires the JNDI <code>InitialContext</code> , it uses the <code>user</code> and <code>password</code> (or <code>encrypted-password</code>) settings.
work-manager	Name of a work manager, configured in the Oracle CEP server <code>config.xml</code> file. This name corresponds to the value of the <code>name</code> child element of the <code>work-manager</code> element in <code>config.xml</code> . 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 <code>work-manager</code> you associate with this JMS inbound adapter: <ul style="list-style-type: none"> ▪ 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. ▪ 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. The default value is the work manager configured for the application itself. For more information, see Section F.44, "work-manager" .

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>
```

7.6.2 JMS Outbound Adapter Component Configuration

Table 7–2 lists the `jms-adapter` element child elements applicable to the JMS outbound adapter.

Table 7–2 *jms-adapter Outbound Component Configuration Child Elements*

Child Element	Description
<code>connection-jndi-name</code>	Optional. The JNDI name of the JMS connection factory. Default value is <code>weblogic.jms.ConnectionFactory</code> , for Oracle CEP server JMS.
<code>connection-password</code> <code>connection-encrypted-password</code>	Optional. Either the password, or encrypted password, for <code>connection-user</code> . Note: Specify either <code>connection-password</code> or <code>connection-encrypted-password</code> , but not both. See Section 7.4, "Encrypting Passwords in the JMS Adapter Component Configuration File" for details on encrypting the password.
<code>connection-user</code>	Optional. When Oracle CEP calls the <code>createConnection</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 CEP uses the <code>user</code> and <code>password</code> (or <code>encrypted-password</code>) settings. 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.
<code>delivery-mode</code>	Specifies the delivery mode: <code>persistent</code> (default value) or <code>nonpersistent</code> .
<code>destination-jndi-name</code>	Required. The JNDI name of the JMS destination. Note: Specify either <code>destination-jndi-name</code> or <code>destination-name</code> , but not both.
<code>destination-name</code>	Required. The actual name of the JMS destination. Note: Specify either <code>destination-jndi-name</code> or <code>destination-name</code> , but not both.
<code>event-type</code>	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 For more information, see Section 7.3, "Creating a Custom Converter Between JMS Messages and Event Types".
<code>jndi-factory</code>	Optional. The JNDI factory name. Default value is <code>weblogic.jndi.WLInitialContextFactory</code> , for Oracle CEP server JMS.
<code>jndi-provider-url</code>	Required. The URL of the JNDI provider.
<code>password</code> <code>encrypted-password</code>	Required. Either the password, or encrypted password, for <code>user</code> . Note: Specify either <code>password</code> or <code>encrypted-password</code> , but not both. See Section 7.4, "Encrypting Passwords in the JMS Adapter Component Configuration File" for details on encrypting the password.
<code>session-transacted</code>	Boolean value that specifies whether or not the session is transactional. If the session is transacted then do not specify <code>session-ack-mode-name</code> . Default: <code>False</code> .

Table 7–2 (Cont.) jms-adapter Outbound Component Configuration Child Elements

Child Element	Description
user	Required. When Oracle CEP 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>
```

Configuring HTTP Publish-Subscribe Server Adapters

This section contains information on the following subjects:

- [Section 8.1, "Overview of HTTP Publish-Subscribe Server Adapter Configuration"](#)
- [Section 8.2, "Configuring an HTTP Pub-Sub Adapter"](#)
- [Section 8.4, "Configuring the HTTP Pub-Sub Adapter EPN Assembly File"](#)
- [Section 8.5, "Configuring the HTTP Pub-Sub Adapter Component Configuration File"](#)

8.1 Overview of HTTP Publish-Subscribe Server Adapter Configuration

An HTTP Publish-Subscribe server (pub-sub server) is a mechanism whereby Web clients, such as browser-based clients, subscribe to channels, receive messages as they become available, and publish messages to these channels, all using asynchronous messages over HTTP. A channel is similar to a JMS topic.

Every instance of Oracle CEP includes a pub-sub server that programmers can use to implement HTTP publish-subscribe functionality in their applications. The pub-sub server is configured in the `config.xml` file along with other server services such as Jetty and JDBC datasources. The pub-sub server is based on the Bayeux protocol (see <http://svn.xantus.org/shortbus/trunk/bayeux/bayeux.html>) proposed by the cometd project (see <http://cometd.com/>). The Bayeux protocol defines a contract between the client and the server for communicating with asynchronous messages over HTTP.

In Oracle CEP, programmers access HTTP publish-subscribe functionality by using the following built-in HTTP publish-subscribe adapters (pub-sub adapters):

- Publishing to a channel: see [Section 8.1.1, "Overview of the Built-In Pub-Sub Adapter for Publishing"](#)
 - Local publishing to a channel: see [Section 8.1.1.1, "Local Publishing"](#).
 - Remote publishing to a channel: see [Section 8.1.1.2, "Remote Publishing"](#).
- Subscribing to a channel: see [Section 8.1.2, "Overview of the Built-In Pub-Sub Adapter for Subscribing"](#).

Oracle CEP also provides a pub-sub API for programmers to create their own custom pub-sub adapters for publishing and subscribing to a channel, if the built-in pub-sub adapters are not adequate. For example, programmers might want to filter incoming messages from a subscribed channel, dynamically create or destroy local channels, and so on. The built-in pub-sub adapters do not provide this functionality, which is why

programmers must implement their own custom pub-sub adapters in this case. For details, see [Chapter 14, "Configuring Custom Adapters."](#)

By default, Oracle CEP performs automatic conversion to and from Oracle CEP event types. Alternatively, you can create a custom converter. See [Section 8.3, "Creating a Custom Converter Between the HTTP Pub-Sub Messages and Event Types"](#).

Oracle CEP can also automatically convert between JSON messages and Oracle CEP event types. See [Section 8.1.3, "Converting Between JSON Messages and Event Types"](#).

The built-in pub-sub adapters work like any other adapter: they are stages in the event processing network, they are defined in the EPN assembly file, and they are configured with the standard component configuration files. Typical configuration options include specifying channels, specifying the local or remote pub-sub server, and user authentication.

The pub-sub server can communicate with any client that can understand the Bayeux protocol. Programmers develop their Web clients using one of the following frameworks:

- Dojo JavaScript library (see <http://dojotoolkit.org/>) that supports the Bayeux protocol. Oracle CEP does not provide this library.
- WebLogic Workshop Flex plug-in that enables development of a Flex client that uses the Bayeux protocol to communicate with a pub-sub server.

For information on securing an HTTP pub-sub server channel, see "Configuring HTTP Publish-Subscribe Server Channel Security" in the *Oracle Complex Event Processing Administrator's Guide*.

8.1.1 Overview of the Built-In Pub-Sub Adapter for Publishing

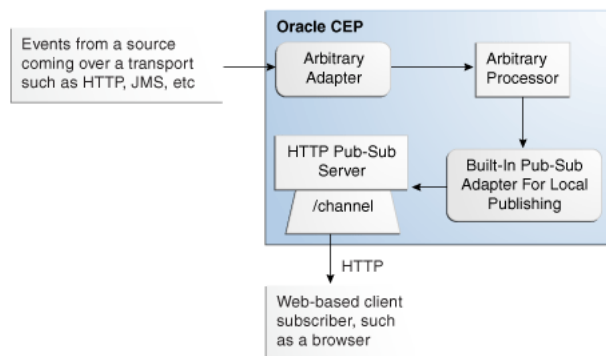
You can use the built-in pub-sub adapter for publishing events to a channel. The built-in pub-sub adapter supports the following publishing modes:

- [Section 8.1.1.1, "Local Publishing"](#)
- [Section 8.1.1.2, "Remote Publishing"](#)

For more information, see [Section 8.1, "Overview of HTTP Publish-Subscribe Server Adapter Configuration"](#).

8.1.1.1 Local Publishing

[Figure 8–1](#) 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.

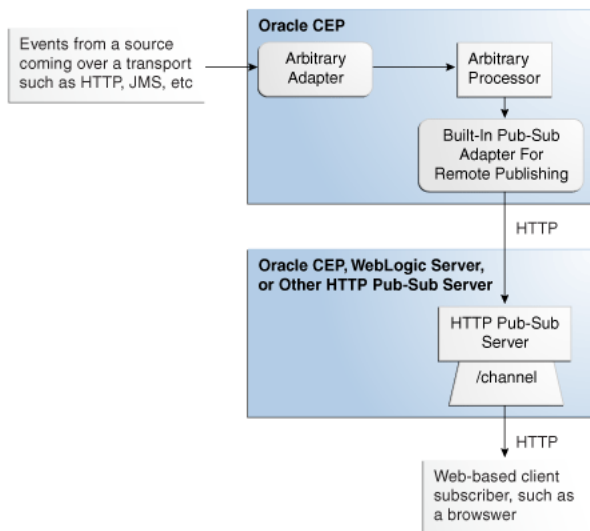
Figure 8–1 Built-In Pub-Sub Adapter For Local Publishing

Note the following in [Figure 8–1](#):

- Events flow from some source into an adapter of an application running in Oracle CEP. This adapter is not required, it is shown only as an example.
- The events flow from the adapter to an arbitrary processor; again, this processor is not required.
- The processor sends the events to the built-in pub-sub adapter for local publishing. The adapter in turn sends the events to the local HTTP pub-sub server configured for the Oracle CEP instance on which the application is deployed. The pub-sub adapter sends the messages to the channel for which it has been configured.
- The local HTTP pub-sub server configured for Oracle CEP then sends the event as a message to all subscribers of the local channel.

8.1.1.2 Remote Publishing

[Figure 8–2](#) shows how the built-in pub-sub adapter for remote publishing fits into a simple event processing network.

Figure 8–2 Built-In Pub-Sub Adapter For Remote Publishing

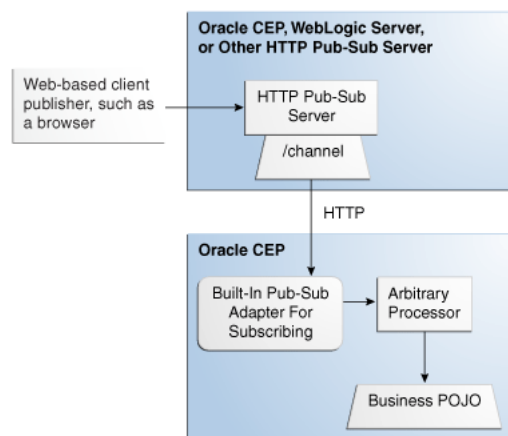
Note the following in [Figure 8–2](#):

- Events flow from some source into an adapter of an application running in Oracle CEP. The arbitrary adapter is not required, it is shown only as an example.
- The events flow from the adapter to an arbitrary processor; again, this processor is not required.
- The processor sends the events to the built-in pub-sub adapter for remote publishing. The adapter in turn sends the events as messages to the remote HTTP pub-sub server for which the adapter is configured; this HTTP pub-sub server could be on another Oracle CEP instance, a WebLogic Server instance, or any other third-party implementation. The pub-sub adapter sends the messages to the channel for which it has been configured.
- The remote HTTP pub-sub server then sends the message to all subscribers of the channel.

8.1.2 Overview of the Built-In Pub-Sub Adapter for Subscribing

Figure 8–3 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 8–3 Built-In Pub-Sub Adapter For Subscribing



Note the following in Figure 8–3:

- Messages are published to a remote HTTP pub-sub server, which could be another instance of Oracle CEP, WebLogic Server, or a third-party implementation. The messages are typically published by Web based clients (shown in graphic), by the HTTP pub-sub server itself, or another server application.
- The built-in pub-sub adapter running in an Oracle CEP application subscribes to the HTTP pub-sub server and receives messages from the specified channel. The adapter converts the messages into the event type configured for the adapter.
- The pub-sub adapter sends the events to a processor. This processor is not required, it is shown only as an example of a typical Oracle CEP application.
- The processor sends the events to a business POJO. Again, this business POJO is not required.

For more information, see [Section 8.1, "Overview of HTTP Publish-Subscribe Server Adapter Configuration"](#).

8.1.3 Converting Between JSON Messages and Event Types

Oracle CEP 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 8.3, "Creating a Custom Converter Between the HTTP Pub-Sub Messages and Event Types"](#) for details.

If you do *not* provide your own converter class, and instead let Oracle CEP take care of the conversion between messages and event types, the following is true:

- You must specify an event type that Oracle CEP uses in its conversion. See [Section 8.2.2, "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/>.

8.2 Configuring an HTTP Pub-Sub Adapter

This section describes how to configure Oracle CEP HTTP pub-sub adapter for both publishing and subscribing:

- [Section 8.2.1, "How to Configure an HTTP Pub-Sub Adapter Using the Oracle CEP IDE for Eclipse"](#)
- [Section 8.2.2, "How to Configure an HTTP Pub-Sub Adapter Manually"](#)

Note: This section assumes that you have already created an Oracle CEP application, along with its EPN assembly file and component configuration files, and that you want to update the application to use the built-in pub-sub adapters. If this is not true, refer to [Chapter 1, "Overview of Creating Oracle CEP Applications"](#) for general information about creating an Oracle CEP application.

8.2.1 How to Configure an HTTP Pub-Sub Adapter Using the Oracle CEP IDE for Eclipse

The simplest way to create and configure an HTTP pub-sub adapter is using the Oracle CEP IDE for Eclipse adapter wizard.

For more information, see [Section 6.4.1.2, "How to Create an Adapter Node"](#).

After using the adapter wizard to create and specify the basic HTTP pub-sub adapter configuration, review [Section 8.2.2, "How to Configure an HTTP Pub-Sub Adapter Manually"](#) to complete the configuration.

8.2.2 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 CEP IDE for Eclipse adapter wizard as [Section 8.2.1, "How to Configure an HTTP Pub-Sub Adapter Using the Oracle CEP 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 1.1.5, "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 B.2, "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 CEP knows to which particular adapter in the EPN assembly file this adapter configuration applies.

For example, assume your configuration file already contains a processor (contents removed for simplicity) and you want to configure instances of each of the three built-in pub-sub adapters; then the updated file might look like the following; details of the adapter configuration will be added in later steps:

```
<?xml 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 CEP application will publish or subscribe, respectively. The remote pub-sub server could be another instance of Oracle CEP, or a WebLogic Server instance, or it could be any third-party HTTP pub-sub server. For example:

```

<http-pub-sub-adapter>
  <name>remotePublisher</name>
  <server-url>http://myhost.com:9102/pubsub</server-url>
  ...
</http-pub-sub-adapter>

```

In the example, the URL of the remote HTTP pub-sub server to which the `remotePublisher` adapter will publish events is `http://myhost.com:9102/pubsub`.

- For each *local* pub-sub adapter for publishing, add a `server-context-path` element to specify the path of the local HTTP pub-sub server associated with the Oracle CEP instance hosting the current Oracle CEP application.

By default, each Oracle CEP server is configured with an HTTP pub-sub server with path `/pubsub`; if, however, you have created a new local HTTP pub-sub server, or changed the default configuration, then specify the value of the `path` child element of the `http-pubsub` element in the server's `config.xml` file. For example:

```

<http-pub-sub-adapter>
  <name>localPublisher</name>
  <server-context-path>/pubsub</server-context-path>
  ...
</http-pub-sub-adapter>

```

- 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 CEP uses the incoming key-value pairs in the message to map the message data to the specified event type.

You can also optionally use the `event-type` element in a pub-sub adapter for publishing if you want to limit the types of events that are published to just those specified by the `event-type` elements. Otherwise, all events sent to the pub-sub adapter are published. For example:

```
<http-pub-sub-adapter>
  <name>remoteSubscriber</name>
  <server-url>http://myhost.com:9102/pubsub</server-url>
  <channel>/channel3</channel>
  <event-type>com.mycompany.httppubsub.PubsubEvent</event-type>
</http-pub-sub-adapter>
```

Be sure this event type has been registered in the EPN assembly file by specifying it as a child element of the `wlevs:event-type-repository` element.

7. Finally, if the HTTP pub-sub server to which the Oracle CEP application is publishing requires user authentication, add `user` and `password` (or `encrypted-password`) elements to specify the username and password or encrypted password. For example:

```
<http-pub-sub-adapter>
  <name>remotePublisher</name>
  <server-url>http://myhost.com:9102/pubsub</server-url>
  <channel>/channel1</channel>
  <event-type>com.mycompany.httppubsub.PubsubEvent</event-type>
  <user>wlevs</user>
  <password>wlevs</password>
</http-pub-sub-adapter>
```

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 CEP make the conversion based on mapping property names between the messages and a specified event type.

See [Section 8.3, "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 CEP instance for local publishing, use Visualizer, the Oracle CEP Administration Tool, to add new channels with the channel pattern required by your application.

For details, see "How to Configure Security for an HTTP Publish-Subscribe Channel" in the *Oracle Complex Event Processing Visualizer User's Guide*.

10. Update the EPN assembly file, adding declarations for each built-in pub-sub adapter you are adding to your application.

See [Section 8.4, "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 24.2.2.1, "Creating the MANIFEST.MF File"](#) for additional information on the manifest file.

8.3 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 Complex 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 `OutboundMessageConverter`; 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>

8.4 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 CEP application define where in the EPN the HTTP pub-sub adapters fit in.

This section describes:

- [Section 8.4.1, "HTTP Pub-Sub Adapter for Publishing EPN Assembly File Configuration"](#)
- [Section 8.4.2, "HTTP Pub-Sub Adapter for Subscribing EPN Assembly File Configuration"](#)

For more information, see:

- [Section 1.1.5, "Component Configuration Files."](#)
- [Section B.2, "Component Configuration Schema `wlevs_application_config.xsd`"](#)

8.4.1 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 CEP application define where in the network the pub-sub adapters fit in.

Also be sure that the event types used by the pub-sub adapters have been registered in the event type repository using the `wlevs:event-type-repository` element.

The following sample EPN file shows an event processing network with two built-in pub-sub adapters for publishing both local and remote publishing); see the text after the example for an explanation:

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



```

http://www.bea.com/ns/wlevs/spring
http://www.bea.com/ns/wlevs/spring/spring-wlevs.xsd">
  <wlevs:event-type-repository>
    <wlevs:event-type type-name="com.mycompany.httppubsub.PubsubEvent">
      <wlevs:class>com.mycompany.httppubsub.PubsubEvent</wlevs:class>
    </wlevs:event-type>
  </wlevs:event-type-repository>
  <wlevs:adapter id="receiveFromFeed"
    class="com.mycompany.httppubsub.ReceiveFromFeed">
  </wlevs:adapter>
  <wlevs:processor id="pubsubProcessor" />
  <wlevs:adapter id="remotePublisher" provider="httppub"/>
  <wlevs:adapter id="localPublisher" provider="httppub"/>
  <wlevs:channel id="feed2processor">
    <wlevs:source ref="receiveFromFeed" />
    <wlevs:listener ref="pubsubProcessor" />
  </wlevs:channel>
  <wlevs:channel id="pubsubStream">
    <wlevs:listener ref="remotePublisher" />
    <wlevs:listener ref="localPublisher" />
    <wlevs:source ref="pubsubProcessor" />
  </wlevs:channel>
</beans>

```

In the preceding example:

- The `receiveFromFeed` adapter is a custom adapter that receives data from some data feed; the details of this adapter are not pertinent to this topic. The `receiveFromFeed` adapter then sends its events to the `pubsubProcessor` via the `feed2processor` channel.
- The `pubsubProcessor` processes the events from the `receiveFromFeed` adapter and then sends them to the `pubsubStream` channel, which in turn sends them to the two built-in pub-sub adapters: `remotePublisher` and `localPublisher`.
- Based on the configuration of these two pub-sub adapters (see examples in [Section 8.2.2, "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 CEP server on which the application is running. The local pub-sub server's path is `/pubsub` and the channel to which the adapter publishes is called `/channel2`.

The following sample EPN file shows an event processing network with one built-in pub-sub adapter for subscribing; see the text after the example for an explanation:

```

<?xml 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 8.2.2, "How to Configure an HTTP Pub-Sub Adapter Manually"](#)), `remoteSubscriber` subscribes to a channel called `/channel13` configured for the remote HTTP pub-sub server hosted at `http://myhost.com:9102/pubsub`. Oracle CEP converts each messages it receives from this channel to an instance of `com.mycompany.httppubsub.PubsubEvent` and then sends it a Spring bean called `myEventBean`.
- The `myEventBean` processes the event as described by the `com.mycompany.httppubsub.MyEventBean` class, and then passes it the `mySink` event source via the `pubsubStream` channel. This section does not discuss the details of these components because they are not pertinent to the HTTP pub-sub adapter topic.

8.4.2 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>

```

8.5 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 8.5.1, "HTTP Pub-Sub Adapter for Publishing Component Configuration"](#)
- [Section 8.5.2, "HTTP Pub-Sub Adapter for Subscribing Component Configuration"](#)

For more information, see:

- [Section 1.1.5, "Component Configuration Files."](#)
- [Section B.2, "Component Configuration Schema `wlevs_application_config.xsd`"](#)

8.5.1 HTTP Pub-Sub Adapter for Publishing Component Configuration

[Table 8–1](#) lists the `http-pub-sub-adapter` element child elements applicable to an HTTP pub-sub adapter for publishing.

Table 8–1 *http-pub-sub-adapter for Publishing Component Configuration Child Elements*

Child Element	Description
<code>server-context-path</code>	<p>Required. For each <i>local</i> HTTP pub-sub adapter for publishing, specify the value of the Oracle CEP 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 CEP instance hosting the current Oracle CEP 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>NOTE: Do not specify this option for a remote HTTP pub-sub adapter.</p>

Table 8–1 (Cont.) http-pub-sub-adapter for Publishing Component Configuration Child Elements

Child Element	Description
server-url	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 CEP application will publish. The remote HTTP pub-sub server could be another instance of Oracle CEP, or a WebLogic Server instance, or it could be any third-party HTTP pub-sub server. For example: http://myhost.com:9102/pubsub NOTE: Do not specify this option for a local HTTP pub-sub adapter.
channel	Required. For both local and remote HTTP pub-sub adapters for publishing, specify the channel that the HTTP pub-sub adapter publishes to.
event-type	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. 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 2.2, "Creating an Oracle CEP Event Type as a JavaBean" .
user	Optional. For both local and remote HTTP pub-sub adapters for publishing, if the HTTP pub-sub server to which the Oracle CEP application is publishing requires user authentication, specify a user name.
password	Optional. For both local and remote HTTP pub-sub adapters for publishing, if the HTTP pub-sub server to which the Oracle CEP application is publishing requires user authentication, specify a password. Choose either <code>password</code> or <code>encrypted-password</code> but not both.
encrypted-password	Optional. For both local and remote HTTP pub-sub adapters for publishing, if the HTTP pub-sub server to which the Oracle CEP 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 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>
</n1:config>
```

```
</nl:config>
```

8.5.2 HTTP Pub-Sub Adapter for Subscribing Component Configuration

Table 8–2 lists the `http-pub-sub-adapter` element child elements applicable to an HTTP pub-sub adapter for subscribing.

Table 8–2 *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 CEP application will publish. The remote HTTP pub-sub server could be another instance of Oracle CEP, or a WebLogic Server instance, or it could be any third-party HTTP pub-sub server NOTE: 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 CEP 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 2.2, "Creating an Oracle CEP Event Type as a JavaBean" .
<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 CEP 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 CEP 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 CEP 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>
  <name>remoteSubscriber</name>
  <server-url>http://myhost.com:9102/pubsub</server-url>
  <channel>/channel3</channel>
  <event-type>com.mycompany.httppubsub.PubsubEvent</event-type>
</http-pub-sub-adapter>
</n1:config>
```

Configuring Channels

This section contains information on the following subjects:

- [Section 9.1, "Overview of Channel Configuration"](#)
- [Section 9.2, "Configuring a Channel"](#)
- [Section 9.3, "Example Channel Configuration Files"](#)

9.1 Overview of Channel Configuration

An Oracle CEP application contains one or more channel components. A channel represents the physical conduit through which events flow between other types of components, such as between adapters and processors, and between processors and event beans (business logic POJOs).

You may use a channel with both streams and relations. For more information, see [Section 9.1.2, "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 C.10, "wlevs:channel"](#).

The default channel configuration is typically adequate for most applications.

However, if you want to change this configuration, you must create a `channel` element in a component configuration file. In this `channel` element, you can specify channel configuration that overrides the defaults.

The component configuration file `channel` element's name element must match the EPN assembly file `channel` element's `id` attribute. For example, given the EPN assembly file `channel` element shown in [Example 9-1](#), the corresponding component configuration file `channel` element is shown in [Example 9-2](#).

Example 9-1 EPN Assembly File Channel Id: priceStream

```
<wlevs:channel id="priceStream" event-type="PriceEvent">
  <wlevs:listener ref="filterFanoutProcessor" />
  <wlevs:source ref="PriceAdapter" />
</wlevs:channel>
```

Example 9-2 Component Configuration File Channel Name: priceStream

```
<channel>
  <name>priceStream</name>
  <max-size>10000</max-size>
  <max-threads>4</max-threads>
</channel>
```

You can create a `channel` element in any of the following component configuration files:

- The default Oracle CEP application configuration file (by default, `META-INF/wlevs/config.xml`).
- A separate configuration file.

If your application has more than one channel, you can create a channel element for each of them in the default `config.xml` file, you can create separate XML files in `META-INF/wlevs` for each, or create a single XML file in `META-INF/wlevs` that contains the configuration for all channels, or even all components of your application (adapters, processors, and channels). Choose the method that best suits your development environment.

By default, Oracle CEP IDE for Eclipse creates one component configuration file and one EPN assembly file.

Component configuration files are deployed as part of the Oracle CEP application bundle. You can later update this configuration at runtime using Oracle CEP Visualizer, the `wlevs.Admin` utility, or manipulating the appropriate JMX Mbeans directly.

This section describes:

- [Section 9.1.1, "When to Use a Channel"](#)
- [Section 9.1.2, "Channels Representing Streams and Relations"](#)
- [Section 9.1.3, "System-Timestamped Channels"](#)
- [Section 9.1.4, "Application-Timestamped Channels"](#)
- [Section 9.1.5, "Controlling Which Queries Output to a Downstream Channel: selector"](#)
- [Section 9.1.6, "Batch Processing Channels"](#)
- [Section 9.1.7, "EventPartitioner Channels"](#)

For more information, see:

- [Section 1.1.5, "Component Configuration Files"](#)
- [Section 4.3, "Creating EPN Assembly Files"](#)
- *Oracle Complex Event Processing Visualizer User's Guide*
- "wlevs.Admin Command-Line Reference" in the *Oracle Complex Event Processing Administrator's Guide*
- "Configuring JMX for Oracle CEP" in the *Oracle Complex Event Processing Administrator's Guide*

9.1.1 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 9.1.7, "EventPartitioner Channels"](#)
- [Table C-9, "Attributes of the `wlevs:channel` Application Assembly Element"](#)

9.1.2 Channels Representing Streams and Relations

A channel can represent either a stream or a relation.

For more information, see:

- [Section 1.1.3.2, "Stream and Relation Sources"](#)
- [Section 9.1, "Overview of Channel Configuration"](#)

9.1.2.1 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).

9.1.2.2 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 9-3](#) shows.

Example 9-3 Channel as Relation: primary-key Attribute

```
...
<wlevs:channel id="priceStream" event-type="PriceEvent" primary-key="stock,broker">
  <wlevs:listener ref="filterFanoutProcessor" />
  <wlevs:source ref="PriceAdapter" />
</wlevs:channel>
```

```
</wlevs:channel>
...
```

Example 9–4 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".

9.1.3 System-Timestamped Channels

By default, channels are system-timestamped. In this case, Oracle CEP will assign a new time from the CPU clock under two conditions: when a new event arrives, and when the configurable heartbeat timeout expires.

For more information, see:

- [Section 9.2.1, "How to Configure a System-Timestamped Channel Using Oracle CEP IDE for Eclipse"](#)
- [Section D.46, "heartbeat"](#)

9.1.4 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 9.2.2, "How to Configure an Application-Timestamped Channel Using Oracle CEP IDE for Eclipse"](#)
- [Section C.3, "wlevs:application-timestamped"](#)
- [Section C.14, "wlevs:expression"](#)

9.1.5 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.

Figure 9–1 shows an EPN with channel `filteredStream` connected to up-stream Oracle CQL processor `filterFanoutProcessor`.

Figure 9–1 EPN With Oracle CQL Processor and Down-Stream Channel



Example 9–5 shows the queries configured for the Oracle CQL processor.

Example 9–5 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 9–5 shows, then, by default, all query results are output to the processor’s out-bound channel (`filteredStream` in Figure 9–1). Optionally, in the component configuration source, you can use the `channel` element `selector` 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 9–6 shows. In this example, query results for query `Yr3Sector` and `Yr2Sector` are output to `filteredStream` but not query results for query `Yr1Sector`.

Example 9–6 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 10, "Configuring Oracle CQL Processors"](#).

For more information, see [Appendix D.78, "selector"](#).

9.1.6 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 9–7](#) shows.

Example 9–7 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 1.1.3.2, "Stream and Relation Sources"](#)
- [Section 1.1.3.3, "Stream and Relation Sinks"](#)
- `batching` in [Table C–9, "Attributes of the `wlevs:channel` Application Assembly Element"](#)

9.1.7 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 22.2.1, "EventPartitioner"](#).

9.2 Configuring a Channel

You can configure a channel manually or by using the Oracle CEP IDE for Eclipse.

See [Section B.2, "Component Configuration Schema `wlevs_application_config.xsd`"](#) for the complete XSD Schema that describes the channel component configuration file.

See [Section 9.3, "Example Channel Configuration Files"](#) for a complete example of a channel configuration file.

This section describes the following topics:

- [Section 9.2.1, "How to Configure a System-Timestamped Channel Using Oracle CEP IDE for Eclipse"](#)
- [Section 9.2.2, "How to Configure an Application-Timestamped Channel Using Oracle CEP IDE for Eclipse"](#)
- [Section 9.2.3, "How to Create a Channel Component Configuration File Manually"](#)

9.2.1 How to Configure a System-Timestamped Channel Using Oracle CEP IDE for Eclipse

This section describes how to create a system-timestamped channel.

The most efficient and least error-prone way to create and edit a channel configuration in the default component configuration file is to use the Oracle CEP IDE for Eclipse. Optionally, you can create a channel configuration file manually (see [Section 9.2.3, "How to Create a Channel Component Configuration File Manually"](#)).

For more information, see:

- [Section 9.1.3, "System-Timestamped Channels"](#)
- [Chapter 6, "Oracle CEP IDE for Eclipse and the Event Processing Network"](#)

To configure a channel using Oracle CEP IDE for Eclipse:

1. Use Oracle CEP IDE for Eclipse to create a channel.
See [Section 6.4.1.1, "How to Create a Basic Node"](#).
2. Optionally, override the default channel assembly file configuration by adding additional `wlevs:channel` attributes and child elements:
 - a. Right-click the channel node and select **Go To Assembly Source**.
 - b. Add the appropriate `wlevs:channel` attributes.
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 C.17, "wlevs:instance-property"](#)
 - [Appendix C.18, "wlevs:listener"](#)
 - [Appendix C.21, "wlevs:property"](#)
 - [Appendix C.22, "wlevs:source"](#)
3. In the Project Explorer, expand your `META-INF/wlevs` directory.
4. Choose the component configuration file you want to use:
 - a. To use the default component configuration file, right-click the `META-INF/wlevs/config.xml` file and select **Open With > XML Editor**.
The file opens in an XML Editor.
 - b. To create a new component configuration file:
 - Right-click the `wlevs` directory and select **New > File**.
The New File dialog appears.
 - Enter a file name.

You can name the file anything you want but the name of the file must end in `.xml`.

- Click **Finish**.

Oracle CEP IDE for Eclipse adds the component configuration file to the `wlevs` directory.

5. Right-click the component configuration file you chose to use and select **Open With > XML Editor**.

The file opens in an XML Editor.

6. If you created a new component configuration file, add the header and `config` element shown in [Example 9-8](#). Otherwise, proceed to step 7.

Example 9-8 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 9-9](#) shows.

Example 9-9 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 9-10](#), the corresponding configuration file `channel` element is shown in [Example 9-11](#).

Example 9-10 EPN Assembly File Channel Id: priceStream

```
<wlevs:channel id="priceStream" event-type="PriceEvent">
    <wlevs:listener ref="filterFanoutProcessor" />
    <wlevs:source ref="PriceAdapter" />
</wlevs:channel>
```

Example 9-11 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 CEP server uses to process events for this channel.

Setting this value has no effect when `max-size` is 0. The default value is 0.

```
<channel>
  <name>priceStream</name>
  <max-threads>2</size>
</channel>
```

When set to 0, the channel acts as a pass-through. When `max-threads > 0`, the channel acts as classic blocking queue, where upstream components are producers of events and the downstream components are the consumers of events. The queue size is defined by the configuration `max-size`. There will be up to `max-threads` number of threads consuming events from the queue.

- Add a `max-size` child element to specify the maximum size of the channel.

Zero-size channels synchronously pass-through events.

Non-zero size channels process events asynchronously, buffering events by the requested size. The default value is 0.

```
<channel>
  <name>priceStream</name>
  <max-size>10000</size>
</channel>
```

- Add a `heartbeat` child element to specify the number of nanoseconds a channel can be idle before Oracle CEP generates a heartbeat event to advance time.

The `heartbeat` child element applies to system-timestamped relations or streams only when no events arrive in the event channels that are feeding the processors and the processor has been configured with a statement that includes some temporal operator, such as a time-based window or a pattern matching with duration.

```
<channel>
  <name>MatchOutputChannel</name>
  <heartbeat>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 9.1.5, "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 10, "Configuring Oracle CQL Processors"](#).

10. Select **File > Save**.

The EPN Editor adds a configuration badge to the channel as [Figure 9–2](#) shows. For more information, see [Section 6.2.7, "Configuration Badging"](#).

Figure 9–2 Channel With Configuration Badge



9.2.2 How to Configure an Application-Timestamped Channel Using Oracle CEP IDE for Eclipse

This section describes how to create an application-timestamped channel.

The most efficient and least error-prone way to create and edit a channel configuration in the default component configuration file is to use the Oracle CEP IDE for Eclipse. Optionally, you can create a channel configuration file manually (see [Section 9.2.3, "How to Create a Channel Component Configuration File Manually"](#)).

For more information, see:

- [Section 9.1.4, "Application-Timestamped Channels"](#)
- [Chapter 6, "Oracle CEP IDE for Eclipse and the Event Processing Network"](#)

To configure a channel using Oracle CEP IDE for Eclipse:

1. Use Oracle CEP IDE for Eclipse to create a channel.
See [Section 6.4.1.1, "How to Create a Basic Node"](#).
2. Optionally, override the default channel assembly file configuration by adding additional `wlevs:channel` attributes and child elements:

- a. Right-click the channel node and select **Go To Assembly Source**.
- b. Add the appropriate `wlevs:channel` attributes.

In particular, specify whether this channel is a stream or relation by configuring attribute `is-relation`:

- To specify this channel as stream, set `is-relation` to `false` (default).
- To specify this channel as a relation, set `is-relation` to `true`.

See [Table 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 CEP uses to generate timestamp values.

Optionally, configure the `wlevs:application-timestamped` attributes:

- `is-total-order`: specifies if the application time published is always strictly greater than the last value used.

Valid values are `true` or `false`. Default: `false`.

For more information, see [Appendix C.3, "wlevs:application-timestamped"](#).

- d. Add other appropriate `wlevs:channel` child elements.
 - [Appendix C.17, "wlevs:instance-property"](#)
 - [Appendix C.18, "wlevs:listener"](#)
 - [Appendix C.21, "wlevs:property"](#)
 - [Appendix C.22, "wlevs:source"](#)
3. In the Project Explorer, expand your `META-INF/wlevs` directory.
4. Choose the component configuration file you want to use:
 - a. To use the default component configuration file, right-click the `META-INF/wlevs/config.xml` file and select **Open With > XML Editor**.
The file opens in an XML Editor.
 - b. To create a new component configuration file:
 - Right-click the `wlevs` directory and select **New > File**.
The New File dialog appears.
 - Enter a file name.
You can name the file anything you want but the name of the file must end in `.xml`.
 - Click **Finish**.
Oracle CEP IDE for Eclipse adds the component configuration file to the `wlevs` directory.
5. Right-click the component configuration file you chose to use and select **Open With > XML Editor**.
The file opens in an XML Editor.
6. If you created a new component configuration file, add the header and `config` element shown in [Example 9–8](#). Otherwise, proceed to step 7.

Example 9–12 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 9–9](#) shows.

Example 9–13 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 9–10](#), the corresponding configuration file channel element is shown in [Example 9–11](#).

Example 9–14 EPN Assembly File Channel Id: priceStream

```

<wlevs:channel id="priceStream" event-type="PriceEvent">
  <wlevs:listener ref="filterFanoutProcessor" />
  <wlevs:source ref="PriceAdapter" />
</wlevs:channel>

```

Example 9–15 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 CEP server uses to process events for this channel.

Setting this value has no effect when `max-size` is 0. The default value is 0.

```

<channel>
  <name>priceStream</name>
  <max-threads>2</size>
</channel>

```

When set to 0, the channel acts as a pass-through. When `max-threads > 0`, the channel acts as classic blocking queue, where upstream components are producers of events and the downstream components are the consumers of events. The queue size is defined by the configuration `max-size`. There will be up to `max-threads` number of threads consuming events from the queue.

- Add a `max-size` child element to specify the maximum size of the channel.

Zero-size channels synchronously pass-through events.

Non-zero size channels process events asynchronously, buffering events by the requested size. The default value is 0.

```

<channel>
  <name>priceStream</name>
  <max-size>10000</size>
</channel>

```

- Add a `selector` child element to specify which up-stream Oracle CQL processor queries are permitted to output their results to the channel.

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 9.1.5, "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 10, "Configuring Oracle CQL Processors"](#).

For more information, [Section D.78, "selector"](#).

10. Select **File > Save**.

The EPN Editor adds a configuration badge to the channel as [Figure 9–2](#) shows. For more information, see [Section 6.2.7, "Configuration Badging"](#).

Figure 9–3 Channel With Configuration Badge



9.2.3 How to Create a Channel Component Configuration File Manually

Although the Oracle CEP IDE for Eclipse is the most efficient and least error-prone way to create and a channel configuration file (see [Section 9.2.1, "How to Configure a System-Timestamped Channel Using Oracle CEP IDE for Eclipse"](#)), alternatively, you can also create and maintain a channel configuration file manually.

For simplicity, the following procedure assumes that you are going to configure all components of an application in a single XML file.

To create a channel component configuration file manually:

1. Create an EPN assembly file and add a `wlevs:channel` element for each channel in your application.

Uniquely identify each `wlevs:channel` with the `id` attribute.

See [Section 4.3, "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 C.3, "wlevs:application-timestamped"](#)
- [Appendix C.17, "wlevs:instance-property"](#)
- [Appendix C.18, "wlevs:listener"](#)
- [Appendix C.21, "wlevs:property"](#)
- [Appendix C.22, "wlevs:source"](#)

3. Create an XML file using your favorite XML editor.

You can name this XML file anything you want, provided it ends with the `.xml` extension.

The root element of the configuration file is `config`, with namespace definitions shown in the next step.

4. For each channel in your application, add a `channel` child element of `config`.

Uniquely identify each channel with the `name` child element. This name must be the same as the value of the `id` attribute in the `channel` element of the EPN assembly file that defines the event processing network of your application. This is how Oracle CEP knows to which particular channel component in the EPN assembly file this channel configuration applies. See [Section 4.3, "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 CEP server uses to process events for this channel.

Setting this value has no effect when `max-size` is 0. The default value is 0.

```
<channel>
  <name>priceStream</name>
  <max-threads>2</size>
</channel>
```

When set to 0, the channel acts as a pass-through. When `max-threads > 0`, the channel acts as classic blocking queue, where upstream components are producers of events and the downstream components are the consumers of events. The queue size is defined by the configuration `max-size`. There will be up to `max-threads` number of threads consuming events from the queue.

- Add a `max-size` child element to specify the maximum size of the channel.

Zero-size channels synchronously pass-through events.

Non-zero size channels process events asynchronously, buffering events by the requested size. The default value is 0.

```
<channel>
  <name>priceStream</name>
  <max-size>10000</size>
</channel>
```

- Add a `heartbeat` child element to specify the number of nanoseconds a channel can be idle before Oracle CEP generates a heartbeat event to advance time.

The `heartbeat` child element applies to system-timestamped relations or streams only when no events arrive in the event channels that are feeding the processors and the processor has been configured with a statement that includes some temporal operator, such as a time-based window or a pattern matching with duration.

```
<channel>
  <name>MatchOutputChannel</name>
  <heartbeat>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 9.1.5, "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 10, "Configuring Oracle CQL Processors"](#).

For more information, [Section D.78, "selector"](#).

6. Save and close the configuration file.

9.3 Example Channel Configuration Files

[Figure 9–4](#) 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 9–4](#)).

Figure 9–4 EPN with Two Channels



This section provides example channel configuration files, including:

- [Section 9.3.1, "Channel Component Configuration File"](#)
- [Section 9.3.2, "Channel EPN Assembly File"](#)

9.3.1 Channel Component Configuration File

[Example 9–16](#) shows a sample component configuration file that configures the two channels shown in [Figure 9–4](#).

Example 9–16 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>
  </channel>
</n1:config>

```

```

        <max-threads>2</max-threads>
    </channel>
</n1:config>

```

9.3.2 Channel EPN Assembly File

Example 9–17 shows a EPN assembly file that configures the two channels shown in Figure 9–4.

Example 9–17 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_3.xsd">

  <wlevs:event-type-repository>
    <wlevs:event-type type-name="PriceEvent">
      <wlevs:properties>
        <wlevs:property name="cusip" type="java.lang.String" />
        <wlevs:property name="bid" type="java.lang.Double" />
        <wlevs:property name="srcId" type="java.lang.String" />
        <wlevs:property name="bidQty" type="java.lang.Integer" />
        <wlevs:property name="ask" type="java.lang.Double" />
        <wlevs:property name="askQty" type="java.lang.Integer" />
        <wlevs:property name="seq" type="java.lang.Long" />
        <wlevs:property name="sector" type="java.lang.String" />
      </wlevs:properties>
    </wlevs:event-type>
    <wlevs:event-type type-name="FilteredPriceEvent">
      <wlevs:properties>
        <wlevs:property name="cusip" type="java.lang.String" />
        <wlevs:property name="bid" type="java.lang.Double" />
        <wlevs:property name="srcId" type="java.lang.String" />
        <wlevs:property name="bidQty" type="java.lang.Integer" />
        <wlevs:property name="ask" type="java.lang.Double" />
        <wlevs:property name="askQty" type="java.lang.Integer" />
        <wlevs:property name="seq" type="java.lang.Long" />
      </wlevs:properties>
    </wlevs:event-type>
    <wlevs:event-type type-name="BidAskEvent">
      <wlevs:properties>
        <wlevs:property name="cusip" type="java.lang.String" />
        <wlevs:property name="bidseq" type="java.lang.Long" />
        <wlevs:property name="bidSrcId" type="java.lang.String" />
        <wlevs:property name="bid" type="java.lang.Double" />
        <wlevs:property name="askseq" type="java.lang.Long" />
        <wlevs:property name="askSrcId" type="java.lang.String" />
        <wlevs:property name="ask" type="java.lang.Double" />
        <wlevs:property name="bidQty" type="java.lang.Integer" />
        <wlevs:property name="askQty" type="java.lang.Integer" />
        <wlevs:property name="intermediateStrategy" type="java.lang.String" />
        <wlevs:property name="correlationId" type="java.lang.Long" />
        <wlevs:property name="priority" type="java.lang.Integer" />
      </wlevs:properties>
    </wlevs:event-type>

```

```

<wlevs:event-type type-name="FinalOrderEvent">
  <wlevs:properties>
    <wlevs:property name="cusip" type="java.lang.String" />
    <wlevs:property name="bidseq" type="java.lang.Long" />
    <wlevs:property name="bidSrcId" type="java.lang.String" />
    <wlevs:property name="bid" type="java.lang.Double" />
    <wlevs:property name="bidQty" type="java.lang.Integer" />
    <wlevs:property name="bidSourceStrategy" type="java.lang.String" />
    <wlevs:property name="askseq" type="java.lang.Long" />
    <wlevs:property name="askSrcId" type="java.lang.String" />
    <wlevs:property name="ask" type="java.lang.Double" />
    <wlevs:property name="askQty" type="java.lang.Integer" />
    <wlevs:property name="askSourceStrategy" type="java.lang.String" />
    <wlevs:property name="correlationId" type="java.lang.Long" />
  </wlevs:properties>
</wlevs:event-type>
</wlevs:event-type-repository>

<!-- Assemble EPN (event processing network) -->
<wlevs:adapter advertise="true" id="PriceAdapter"
  provider="csvgen">
  <wlevs:instance-property name="port" value="9008" />
  <wlevs:instance-property name="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>
```

Configuring Oracle CQL Processors

This section contains information on the following subjects:

- [Section 10.1, "Overview of Oracle CQL Processor Configuration"](#)
- [Section 10.2, "Configuring an Oracle CQL Processor"](#)
- [Section 10.3, "Configuring an Oracle CQL Processor Table Source"](#)
- [Section 10.4, "Configuring an Oracle CQL Processor Cache Source"](#)
- [Section 10.5, "Example Oracle CQL Processor Configuration Files"](#)

Note: Oracle CQL replaces Event Processing Language (EPL) in Oracle CEP 11g Release 1 (11.1.1). Oracle CEP supports EPL for backwards compatibility. For more information, see [Chapter 11, "Configuring EPL Processors"](#).

10.1 Overview of Oracle CQL Processor Configuration

An Oracle CEP application contains one or more complex event processors, or *processors* for short. Each processor takes as input events from one or more adapters; these adapters in turn listen to data feeds that send a continuous stream of data from a source. The source could be anything, from a financial data feed to the Oracle CEP load generator.

The main feature of an Oracle CQL processor is its associated Oracle Continuous Query Language (Oracle CQL) rules that select a subset of the incoming events to then pass on to the component that is listening to the processor. The listening component could be another processor, or the business object POJO that typically defines the end of the event processing network, and thus does something with the events, such as publish them to a client application. For more information on Oracle CQL, see the *Oracle Complex Event Processing CQL Language Reference*.

For each Oracle CQL processor in your application, you must create a `processor` element in a component configuration file. In this `processor` element you specify the initial set of Oracle CQL rules of the processor and any optional processor configuration.

You can configure additional optional Oracle CQL processor features in the Oracle CQL processor EPN assembly file.

The component configuration file `processor` element's name element must match the EPN assembly file `processor` element's `id` attribute. For example, given the EPN assembly file `processor` element shown in [Example 10-1](#), the corresponding component configuration file `processor` element is shown in [Example 10-2](#).

Example 10–1 EPN Assembly File Oracle CQL Processor Id: proc

```
<wlevs:processor id="proc">
  <wlevs:table-source ref="Stock" />
</wlevs:processor>
```

Example 10–2 Component Configuration File Oracle CQL Processor Name: proc

```
<processor>
  <name>proc</name>
  <rules>
    <query id="q1"><![CDATA[
      SELECT ExchangeStream.symbol, ExchangeStream.price, Stock.exchange
      FROM   ExchangeStream [Now], Stock
      WHERE  ExchangeStream.symbol = Stock.symbol
    ]]></query>
  </rules>
</procesor>
```

You can create a `processor` element in any of the following component configuration files:

- The default Oracle CEP application configuration file (by default, `META-INF/wlevs/config.xml`).
- A separate configuration file.

If your application has more than one processor, you can create a `processor` element for each of them in the default `config.xml` file, you can create separate XML files in `META-INF/wlevs` for each, or create a single XML file in `META-INF/wlevs` that contains the configuration for all processors, or even all components of your application (adapters, processors, and channels). Choose the method that best suits your development environment.

By default, Oracle CEP IDE for Eclipse creates one component configuration file and one EPN assembly file. When you create an Oracle CQL processor using Oracle CEP IDE for Eclipse, by default, the processor element is added to the default component configuration file `META-INF/wlevs/config.xml` file. Using Oracle CEP IDE for Eclipse, you can choose to create a new configuration file or use an existing configuration file at the time you create the Oracle CQL processor.

Component configuration files are deployed as part of the Oracle CEP application bundle. You can later update this configuration at runtime using Oracle CEP Visualizer, the `wlevs.Admin` utility, or manipulating the appropriate JMX Mbeans directly.

For more information, see:

- [Section 1.1.5, "Component Configuration Files"](#)
- [Section 1.1.4, "EPN Assembly File"](#)
- [Section 4.3, "Creating EPN Assembly Files"](#)
- *Oracle Complex Event Processing Visualizer User's Guide*
- "wlevs.Admin Command-Line Reference" in the *Oracle Complex Event Processing Administrator's Guide*
- "Configuring JMX for Oracle CEP" in the *Oracle Complex Event Processing Administrator's Guide*

For more information on Oracle CQL processor configuration, see:

- [Section 10.1.1, "Controlling Which Queries Output to a Downstream Channel"](#)

- [Section 10.2, "Configuring an Oracle CQL Processor"](#)
- [Section 10.3, "Configuring an Oracle CQL Processor Table Source"](#)
- [Section 10.4, "Configuring an Oracle CQL Processor Cache Source"](#)
- [Section 10.5, "Example Oracle CQL Processor Configuration Files"](#)

10.1.1 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 9.1.5, "Controlling Which Queries Output to a Downstream Channel: selector"](#).

10.2 Configuring an Oracle CQL Processor

You can configure a processor manually or by using the Oracle CEP IDE for Eclipse.

See [Section B.2, "Component Configuration Schema `wlevs_application_config.xsd`"](#) for the complete XSD Schema that describes the processor component configuration file.

See [Section 10.5, "Example Oracle CQL Processor Configuration Files"](#) for a complete example of an Oracle CQL processor component configuration file and assembly file.

This section describes the following topics:

- [Section 10.2.1, "How to Configure an Oracle CQL Processor Using Oracle CEP IDE for Eclipse"](#)
- [Section 10.2.2, "How to Create an Oracle CQL Processor Component Configuration File Manually"](#)

10.2.1 How to Configure an Oracle CQL Processor Using Oracle CEP IDE for Eclipse

The most efficient and least error-prone way to create and edit a processor is to use the Oracle CEP IDE for Eclipse. Optionally, you can create and edit a processor manually (see [Section 10.2.2, "How to Create an Oracle CQL Processor Component Configuration File Manually"](#)).

To configure an Oracle CQL processor using Oracle CEP IDE for Eclipse:

1. Use Oracle CEP IDE for Eclipse to create a processor.

See [Section 6.4.1.3, "How to Create a Processor Node"](#).

When you use the EPN editor to create an Oracle CQL processor, Oracle CEP IDE for Eclipse prompts you to choose either the default component configuration file or a new component configuration file. For more information, see [Chapter 6, "Oracle CEP IDE for Eclipse and the Event Processing Network"](#).

2. Right-click the processor node and select **Go to Configuration Source**.

Oracle CEP IDE for Eclipse opens the appropriate component configuration file. The default processor component configuration is shown in [Example 10-3](#).

The default processor component configuration includes a `name` element and `rules` element.

Use the `rules` element to group the child elements you create to contain the Oracle CQL statements this processor executes, including:

- `rule`: contains Oracle CQL statements that register or create user-defined windows. The `rule` element `id` attribute must match the name of the window.
- `view`: contains Oracle CQL view statements (the Oracle CQL equivalent of subqueries). The `view` element `id` attribute defines the name of the view.
- `query`: contains Oracle CQL select statements. The `query` element `id` attribute defines the name of the query.

The default processor component configuration includes a dummy `query` element with `id` `Query`.

Example 10-3 Default Processor Component Configuration

```
<processor>
  <name>proc</name>
  <rules>
    <query id="Query"><!-- <![CDATA[ select * from MyChannel [now] ]]> -->
    </query>
  </rules>
</processor>
```

3. Replace the dummy `query` element with the `rule`, `view`, and `query` elements you create to contain the Oracle CQL statements this processor executes.

For more information, see "Introduction to Oracle CQL Queries, Views, and Joins" in the *Oracle Complex Event Processing CQL Language Reference*.

4. Select **File > Save**.
5. Optionally, configure additional Oracle CQL processor features in the assembly file:
 - [Section 10.1.1, "Controlling Which Queries Output to a Downstream Channel"](#)
 - [Section 10.3, "Configuring an Oracle CQL Processor Table Source"](#)
 - [Section 10.4, "Configuring an Oracle CQL Processor Cache Source"](#)

10.2.2 How to Create an Oracle CQL Processor Component Configuration File Manually

Although the most efficient and least error-prone way to create and edit a processor configuration is to use the Oracle CEP IDE for Eclipse (see [Section 10.2.1, "How to Configure an Oracle CQL Processor Using Oracle CEP IDE for Eclipse"](#)), alternatively, you can also create and maintain a processor configuration file manually.

This section describes the main steps to create the processor configuration file manually. For simplicity, it is assumed in the procedure that you are going to configure all processors in a single XML file, although you can also create separate files for each processor.

To create an Oracle CQL processor component configuration file manually:

1. Design the set of Oracle CQL rules that the processor executes. These rules can be as simple as selecting *all* incoming events to restricting the set based on time, property values, and so on, as shown in the following:

```
SELECT *
FROM   TradeStream [Now]
WHERE  price > 10000
```

Oracle CQL is similar in many ways to Structure Query Language (SQL), the language used to query relational database tables, although the syntax between the two differs in many ways. The other big difference is that Oracle CQL queries take another dimension into account (time), and the processor executes the Oracle CQL continually, rather than SQL queries that are static.

For more information, see "Introduction to Oracle CQL Queries, Views, and Joins" in the *Oracle Complex Event Processing CQL Language Reference*.

2. Create the processor configuration XML file that will contain the Oracle CQL rules you designed in the preceding step, as well as other optional features, for each processor in your application.

You can name this XML file anything you want, provided it ends with the `.xml` extension.

The root element of the processor configuration file is `config`, with namespace definitions shown in the next step.

3. For each processor in your application, add a `processor` child element of `config`.

Uniquely identify each processor with the `name` child element. This name must be the same as the value of the `id` attribute in the `wlevs:processor` element of the EPN assembly file that defines the event processing network of your application. This is how Oracle CEP knows to which particular processor component in the EPN assembly file this processor configuration applies. See [Section 4.3, "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 [Now]
        WHERE  price > 10000
      ]]></query>
    </rules>
  </processor>
</n1:config>]]></query>
```

5. Save and close the file.

6. Optionally, configure additional Oracle CQL processor features in the assembly file:

- [Section 10.1.1, "Controlling Which Queries Output to a Downstream Channel"](#)
- [Section 10.3, "Configuring an Oracle CQL Processor Table Source"](#)
- [Section 10.4, "Configuring an Oracle CQL Processor Cache Source"](#)

10.3 Configuring an Oracle CQL Processor Table Source

You can configure an Oracle CQL processor to access a table in a relational database as an event stream in which each row in the table is represented as a tuple.

In this section, assume that you create the table you want to access using the SQL statement that [Example 10–4](#) shows.

Example 10–4 Table Create SQL Statement

```
create table Stock (symbol varchar(16), exchange varchar(16));
```

After configuration, you can define Oracle CQL queries that access the `Stock` table as if it was just another event stream. In the following example, the query joins one event stream `ExchangeStream` with the `Stock` table:

Example 10–5 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 Complex Event Processing CQL Language Reference*.

10.3.1 How to Configure an Oracle CQL Processor Table Source Using Oracle CEP IDE for Eclipse

The most efficient and least error-prone way to configure an Oracle CQL processor to access a relational database table is to use the Oracle CEP IDE for Eclipse.

To configure an Oracle CQL processor table source using Oracle CEP IDE for Eclipse:

1. Create a data source for the database that contains the table you want to use.

[Example 10–6](#) shows an example Oracle CEP server `config.xml` file with data source `StockDS`.

Example 10–6 Oracle CEP Server config.xml File With Data Source StockDS

```
<?xml version="1.0" encoding="UTF-8"?>
<n1:config xsi:schemaLocation="http://www.bea.com/ns/wlevs/config/server wlevs_server_
config.xsd"
  xmlns:n1="http://www.bea.com/ns/wlevs/config/server"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <domain>
    <name>ocep_domain</name>
  </domain>
```

```

...
<data-source>
  <name>StockDs</name>
  <connection-pool-params>
    <initial-capacity>1</initial-capacity>
    <max-capacity>10</max-capacity>
  </connection-pool-params>
  <driver-params>
    <url>jdbc:derby:</url>
    <driver-name>org.apache.derby.jdbc.EmbeddedDriver</driver-name>
    <properties>
      <element>
        <name>databaseName</name>
        <value>db</value>
      </element>
      <element>
        <name>create</name>
        <value>true</value>
      </element>
    </properties>
  </driver-params>
  <data-source-params>
    <jndi-names>
      <element>StockDs</element>
    </jndi-names>
    <global-transactions-protocol>None</global-transactions-protocol>
  </data-source-params>
</data-source>
...
</nl:config>

```

For more information, see "Configuring Access to a Relational Database" in the *Oracle Complex Event Processing Administrator's Guide*.

2. Use Oracle CEP IDE for Eclipse to create a table node.
See [Section 6.4.1.1, "How to Create a Basic Node"](#).
3. Use Oracle CEP IDE for Eclipse to create an Oracle CQL processor.
See [Section 6.4.1.3, "How to Create a Processor Node"](#).
4. Connect the table node to the Oracle CQL processor node.
See [Section 6.4.2.1, "How to Connect Nodes"](#).
The EPN Editor adds a `wlevs:table-source` element to the target processor node that references the source table.
5. Right-click the table node in your EPN and select **Go to Assembly Source**.
Oracle CEP IDE for Eclipse opens the EPN assembly file for this table node.
6. Edit the `table` element as [Example 10–7](#) shows and configure the `table` element attributes as shown in [Table 10–1](#).

Example 10–7 EPN Assembly File table Element

```
<wlevs:table id="Stock" event-type="StockEvent" data-source="StockDs" />
```

Table 10–1 EPN Assembly File table Element Attributes

Attribute	Description
id	The name of the table source. Subsequent references to this table source use this name.
event-type	The type-name you specify for the table event-type you create in step 9.
data-source	The data-source name you specified in the Oracle CEP server config.xml file in step 1.

- Right-click the Oracle CQL processor node connected to the table in your EPN and select **Go to Assembly Source**.

Oracle CEP IDE for Eclipse opens the EPN assembly file for this Oracle CQL processor.

- Edit the Oracle CQL processor element's table-source child element as [Example 10–8](#) shows.

Set the ref attribute to the id of the table element you specified in step 6.

Example 10–8 EPN Assembly File table-source Element

```
<wlevs:processor id="proc">
  <wlevs:table-source ref="Stock" />
</wlevs:processor>
```

- Edit the EPN assembly file to update the event-type-repository element with a new event-type child element for the table as [Example 10–9](#) shows.

Create a property child element for each column of the table you want to access and configure the property attributes as shown in [Table 10–2](#).

Example 10–9 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>
```

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

Attribute	Description
name	The name of the table column you want to access as specified in the SQL create table statement. You do not need to specify all columns.
type	The Oracle CEP Java type from Table 10–3 that corresponds to the column's SQL data type. In Example 10–4 , the type value for column symbol is char[].
length	The column size as specified in the SQL create table statement. In Example 10–4 , the length of column symbol is 16.

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

SQL Type	Oracle CEP Java Type	com.bea.wlevs.ede.api.Type	Description
ARRAY	[Ljava.lang.Object		Array, of depth 1, of java.lang.Object.
BIGINT	java.math.BigInteger	bigint	An instance of java.math.BigInteger.
BINARY	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 ¹ or DOUBLE ²	java.lang.Double	double	An instance of java.lang.Double
BINARY_FLOAT ¹ or FLOAT ²	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 Complex Event Processing CQL Language Reference</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.

¹ Oracle SQL.² Non-Oracle SQL.

For more information on creating event types, see:

- [Chapter 2.1.4, "Creating Oracle CEP Event Types"](#)
- [Section 2.1.3, "Event Type Data Types"](#)

10. Right-click the Oracle CQL processor node connected to the table in your EPN and select **Go to Configuration Source**.

Oracle CEP 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 10–10](#).

Example 10–10 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 Complex Event Processing CQL Language Reference*.

10.4 Configuring an Oracle CQL Processor Cache Source

You can configure an Oracle CQL processor to access the Oracle CEP cache.

For more information, see:

- [Section 12.1, "Overview of Oracle CEP Cache Configuration"](#)
- [Section 12.9, "Accessing a Cache From an Oracle CQL User-Defined Function"](#)
- [Section 12.5, "Accessing a Cache From an Oracle CQL Statement"](#)

10.5 Example Oracle CQL Processor Configuration Files

This section provides example Oracle CQL processor configuration files, including:

- [Section 10.5.1, "Oracle CQL Processor Component Configuration File"](#)
- [Section 10.5.2, "Oracle CQL Processor EPN Assembly File"](#)

10.5.1 Oracle CQL Processor Component Configuration File

The following example shows a component configuration file for an Oracle CQL processor.

```
<?xml version="1.0" encoding="UTF-8"?>
<n1:config
  xsi:schemaLocation="http://www.bea.com/ns/wlevs/config/application wlevs_application_
config.xsd"
  xmlns:n1="http://www.bea.com/ns/wlevs/config/application"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <processor>
    <name>proc</name>
    <rules>
      <view id="lastEvents" schema="cusip bid srcId bidQty ask askQty seq"><![CDATA[
        select mod(price)
        from filteredStream[partition by srcId, cusip rows 1]
```

```

    ]]></view>
    <query id="q1"><![CDATA[
        SELECT *
        FROM   lastEvents [Now]
        WHERE  price > 10000
    ]]></query>
</rules>
</processor>
</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 10.5.2, "Oracle CQL Processor EPN Assembly File"](#)).

This Oracle CQL processor component configuration file also defines a view element to specify an Oracle CQL view statement (the Oracle CQL equivalent of a subquery). The results of the view's select are not output to a down-stream channel.

Finally, this Oracle CQL processor component configuration file defines a `query` element to specify an Oracle CQL query statement. The query statement selects from the view. By default, the results of a query are output to a down-stream channel. You can control this behavior in the channel configuration using a `selector` element. For more information, see:

- [Section 9.2.1, "How to Configure a System-Timestamped Channel Using Oracle CEP IDE for Eclipse"](#)
- [Section 9.2.2, "How to Configure an Application-Timestamped Channel Using Oracle CEP IDE for Eclipse"](#)

10.5.2 Oracle CQL Processor EPN Assembly File

The following example shows an EPN assembly file for an Oracle CQL processor.

```

<?xml 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_3.xsd">

  <wlevs:event-type-repository>
    <wlevs:event-type type-name="ExchangeEvent">
      <wlevs:properties>
        <wlevs:property name="symbol" type="[C" length="16" />
        <wlevs:property name="price" type="java.lang.Double" />
      </wlevs:properties>
    </wlevs:event-type>
    <wlevs:event-type type-name="StockExchangeEvent">
      <wlevs:properties>
        <wlevs:property name="symbol" type="[C" length="16" />
        <wlevs:property name="price" type="java.lang.Double" />
        <wlevs:property name="exchange" type="[C" length="16" />
      </wlevs:properties>
    </wlevs:event-type>
  </wlevs:event-type-repository>

```

```
<wlevs:event-type type-name="StockEvent">
  <wlevs:properties>
    <wlevs:property name="symbol" type="C" length="16" />
    <wlevs:property name="exchange" type="C" length="16" />
  </wlevs:properties>
</wlevs:event-type>
</wlevs:event-type-repository>

<!-- 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 EPL Processors

This section contains information on the following subjects:

- [Section 11.1, "Overview of EPL Processor Component Configuration"](#)
- [Section 11.2, "Configuring an EPL Processor"](#)
- [Section 11.3, "Configuring an EPL Processor Cache Source"](#)
- [Section 11.4, "Example EPL Processor Configuration Files"](#)

Note: Oracle CQL replaces Event Processing Language (EPL) in Oracle CEP 11g Release 1 (11.1.1). Oracle CEP supports EPL for backwards compatibility. For more information, see [Chapter 10, "Configuring Oracle CQL Processors"](#).

11.1 Overview of EPL Processor Component Configuration

An Oracle CEP application contains one or more complex event processors, or *processors* for short. Each processor takes as input events from one or more adapters; these adapters in turn listen to data feeds that send a continuous stream of data from a source. The source could be anything, from a financial data feed to the Oracle CEP load generator.

The main feature of an EPL processor is its associated Event Processing Language (EPL) rules that select a subset of the incoming events to then pass on to the component that is listening to the processor. The listening component could be another processor, or the business object POJO that typically defines the end of the event processing network, and thus does something with the events, such as publish them to a client application. For more information about EPL, see the *Oracle Complex Event Processing EPL Language Reference*.

For each EPL processor in your application, you must create a `processor` element in a component configuration file. In this `processor` element you specify the initial set of EPL rules of the processor and any optional processor configuration such as:

- JDBC datasource reference if your Oracle CEP application requires a connection to a relational database.
- Enable monitoring of the processor.

You can configure additional optional EPL processor features in the EPL processor EPN assembly file.

The component configuration file `processor` element's `name` element must match the EPN assembly file `processor` element's `id` attribute. For example, given the EPN

assembly file processor element shown in [Example 11–1](#), the corresponding component configuration file processor element is shown in [Example 11–2](#).

Example 11–1 EPN Assembly File EPL Processor Id: proc

```
<wlevs:processor id="proc" provider="epl" >
  <wlevs:table-source ref="Stock" />
</wlevs:processor>
```

Example 11–2 Component Configuration File EPL Processor Name: proc

```
<processor>
  <name>proc</name>
  <rules>
    <rule id="myRule"><![CDATA[
      SELECT symbol, AVG(price)
      FROM (SELECT * FROM MarketTrade WHERE blockSize > 10)
      RETAIN 100 EVENTS PARTITION BY symbol WITH LARGEST price
      GROUP BY symbol
      HAVING AVG(price) >= 100
      ORDER BY symbol
    ]]></rule>
  </rules>
</processor>
```

Note: Because Oracle CQL replaces Event Processing Language (EPL) in Oracle CEP 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 `epl` as [Example 11–1](#) shows. Oracle CEP supports EPL for backwards compatibility. For more information, see [Chapter 10, "Configuring Oracle CQL Processors"](#).

You can create a `processor` element in any of the following component configuration files:

- The default Oracle CEP application configuration file (by default, `META-INF/wlevs/config.xml`).
- A separate configuration file.

If your application has more than one processor, you can create a `processor` element for each of them in the default `config.xml` file, you can create separate XML files in `META-INF/wlevs` for each, or create a single XML file in `META-INF/wlevs` that contains the configuration for all processors, or even all components of your application (adapters, processors, and channels). Choose the method that best suits your development environment.

By default, Oracle CEP IDE for Eclipse creates one component configuration file and one EPN assembly file. When you create an EPL processor using Oracle CEP IDE for Eclipse, by default, the processor element is added to the default component configuration file `META-INF/wlevs/config.xml` file.

Component configuration files are deployed as part of the Oracle CEP application bundle. You can later update this configuration at runtime using Oracle CEP Visualizer, the `wlevs.Admin` utility, or manipulating the appropriate JMX Mbeans directly.

For more information, see:

- [Section 1.1.5, "Component Configuration Files"](#)
- [Section 1.1.4, "EPN Assembly File"](#)
- [Section 4.3, "Creating EPN Assembly Files"](#)
- *Oracle Complex Event Processing Visualizer User's Guide*
- "wlevs.Admin Command-Line Reference" in the *Oracle Complex Event Processing Administrator's Guide*
- "Configuring JMX for Oracle CEP" in the *Oracle Complex Event Processing Administrator's Guide*

For more information on EPL processor configuration, see:

- [Section 11.2, "Configuring an EPL Processor"](#)
- [Section 11.3, "Configuring an EPL Processor Cache Source"](#)
- [Section 11.4, "Example EPL Processor Configuration Files"](#)

11.2 Configuring an EPL Processor

This section describes the main steps to create the processor configuration file. For simplicity, it is assumed in the procedure that you are going to configure all processors in a single XML file, although you can also create separate files for each processor.

See [Section B.2, "Component Configuration Schema wlevs_application_config.xsd"](#) for the complete XSD Schema that describes the processor configuration file.

See [Section 11.4, "Example EPL Processor Configuration Files"](#) for a complete example of a processor configuration file.

11.2.1 How to Configure an EPL Processor Manually

You can configure an EPL processor manually using your preferred text editor.

To configure an EPL processor:

1. Design the set of EPL rules that the processor executes. These rules can be as simple as selecting *all* incoming events to restricting the set based on time, property values, and so on, as shown in the following two examples:

```
SELECT * from Withdrawal RETAIN ALL
SELECT symbol, AVG(price)
FROM (SELECT * FROM MarketTrade WHERE blockSize > 10)
RETAIN 100 EVENTS PARTITION BY symbol WITH LARGEST price
GROUP BY symbol
HAVING AVG(price) >= 100
ORDER BY symbol
```

EPL is similar in many ways to Structure Query Language (SQL), the language used to query relational database tables, although the syntax between the two differs in many ways. The other big difference is that EPL queries take another dimension into account (time), and the processor executes the EPL continually, rather than SQL queries that are static.

For additional conceptual information about EPL, and examples and reference information to help you design and write your own EPL rules, see *Oracle Complex Event Processing EPL Language Reference*.

2. Create the processor configuration XML file that will contain the EPL rules you designed in the preceding step, as well as other optional features, for each processor in your application.

You can name this XML file anything you want, provided it ends with the `.xml` extension.

The root element of the processor configuration file is `config`, with namespace definitions shown in the next step.

3. For each processor in your application, add a `processor` child element of `config`.

Uniquely identify each processor with the `name` child element. This name must be the same as the value of the `id` attribute in the `wlevs:processor` element of the EPN assembly file that defines the event processing network of your application. This is how Oracle CEP knows to which particular processor component in the EPN assembly file this processor configuration applies. See [Section 4.3, "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 CEP 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 CEP supports EPL for backwards compatibility. For more information, see [Chapter 10, "Configuring Oracle CQL Processors"](#).

Caution: Identifiers and names in XML files are case sensitive, so be sure you specify the same case when referencing the component's identifier in the EPN assembly file.

4. Add a `rules` child element to each `processor` to group together one or more `rule` elements that correspond to the set of EPL rules you have designed for this processor.

Use the required `id` attribute of the `rule` element to uniquely identify each rule. Use the XML CDATA type to input the actual EPL rule. For example:

```
<processor>
  <name>firstProcessor</name>
  <rules>
    <rule id="myFirstRule"><![CDATA[
      SELECT * from Withdrawal RETAIN ALL
    ]]></rule>
    <rule id="mySecondRule"><![CDATA[
      SELECT * from Checking RETAIN ALL
    ]]></rule>
  </rules>
</processor>
```

5. Optionally, override the default processor configuration by adding additional `processor` child elements:

- Optionally add a `database` child element of the `processor` element to define a JDBC data source for your application. This is required if your EPL rules join a stream of events with an actual relational database table.

Use the `name` child element of `database` to uniquely identify the datasource.

Use the `data-source-name` child element of `database` to specify the actual name of the data source; this name corresponds to the `name` child element of the `data-source` configuration object in the `config.xml` file of your domain.

For more information, see "Configuring Access to a Relational Database" in the *Oracle Complex Event Processing Administrator's Guide*.

For example:

```
<processor>
  <name>firstProcessor</name>
  <rules>
    ....
  </rules>
  <database>
    <name>myDataSource</name>
    <data-source-name>rdbmsDataSource</data-source-name>
  </database>
</processor>
```

6. Save and close the file.
7. Optionally, configure additional EPL processor features in the assembly file:
 - [Section 11.3, "Configuring an EPL Processor Cache Source"](#)

11.3 Configuring an EPL Processor Cache Source

You can configure an EPL processor to access the Oracle CEP cache.

For more information, see:

- [Section 12.1, "Overview of Oracle CEP Cache Configuration"](#)

- [Section 12.10, "Accessing a Cache From an EPL User-Defined Function"](#)
- [Section 12.6, "Accessing a Cache From an EPL Statement"](#)

11.4 Example EPL Processor Configuration Files

This section provides example Oracle CQL processor configuration files, including:

- [Section 11.4.1, "EPL Processor Component Configuration File"](#)
- [Section 11.4.2, "EPL Processor EPN Assembly File"](#)

11.4.1 EPL Processor Component Configuration File

The following example shows how to configure one of the sample EPL queries shown in [Section 11.2, "Configuring an EPL Processor"](#) for the `myProcessor` EPL processor:

```
<?xml version="1.0" encoding="UTF-8"?>
<n1:config xmlns:n1="http://www.bea.com/ns/wlevs/config/application"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <processor>
    <name>myProcessor</name>
    <rules>
      <rule id="myRule"><![CDATA[
        SELECT symbol, AVG(price)
        FROM (SELECT * FROM MarketTrade WHERE blockSize > 10)
        RETAIN 100 EVENTS PARTITION BY symbol WITH LARGEST price
        GROUP BY symbol
        HAVING AVG(price) >= 100
        ORDER BY symbol
      ]]></rule>
    </rules>
  </processor>
</n1:config>
```

In the example, the `name` element specifies that the processor for which the single EPL rule is being configured is called `myProcessor`. This in turn implies that the EPN assembly file that defines your application must include a corresponding `<wlevs:processor id="myProcessor" provider="epl" />` element to link these EPL rules with an actual `myProcessor` EPL processor instance (see [Section 11.4.2, "EPL Processor EPN Assembly File"](#)).

11.4.2 EPL Processor EPN Assembly File

The following example shows an EPN assembly file for an EPL processor.

```
<?xml 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_3.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="epl" />
<wlevs:channel
  id="helloworldOutputChannel" event-type="HelloWorldEvent" advertise="true">
  <wlevs:listener>
    <bean class="com.bea.wlevs.example.helloworld.HelloWorldBean"/>
  </wlevs:listener>
  <wlevs:source ref="helloworldProcessor"/>
</wlevs:channel>
</beans>
```

Configuring Caching

This section contains information on the following subjects:

- [Section 12.1, "Overview of Oracle CEP Cache Configuration"](#)
- [Section 12.2, "Configuring an Oracle CEP Local Caching System and Cache"](#)
- [Section 12.3, "Configuring an Oracle Coherence Caching System and Cache"](#)
- [Section 12.4, "Configuring a Third-Party Caching System and Cache"](#)
- [Section 12.5, "Accessing a Cache From an Oracle CQL Statement"](#)
- [Section 12.6, "Accessing a Cache From an EPL Statement"](#)
- [Section 12.7, "Accessing a Cache From an Adapter"](#)
- [Section 12.8, "Accessing a Cache From a Business POJO"](#)
- [Section 12.9, "Accessing a Cache From an Oracle CQL User-Defined Function"](#)
- [Section 12.10, "Accessing a Cache From an EPL User-Defined Function"](#)
- [Section 12.11, "Accessing a Cache Using JMX"](#)

12.1 Overview of Oracle CEP Cache Configuration

A *cache* is a temporary storage area for events, created exclusively to improve the overall performance of your Oracle CEP application; it is not necessary for the application to function correctly. Oracle CEP applications can optionally publish or consume events to and from a cache to increase the availability of the events and increase the performance of their applications.

A *caching system* refers to a configured instance of a caching implementation. A caching system defines a named set of configured caches as well as the configuration for remote communication if any of the caches are distributed across multiple machines.

Oracle CEP supports the following caching systems:

- Oracle CEP local cache: a local, in-memory single-JVM cache.
- Oracle Coherence: a JCache-compliant in-memory distributed data grid solution for clustered applications and application servers. It coordinates updates to the data using cluster-wide concurrency control, replicates data modifications across the cluster using the highest performing clustered protocol available, and delivers notifications of data modifications to any servers that request them. You take advantage of Oracle Coherence features using the standard Java collections API to access and modify data, and use the standard JavaBean event model to receive data change notifications.

Note: Before you can use Oracle CEP with Oracle Coherence, you must obtain a valid Oracle Coherence license such as a license for Coherence Enterprise Edition, Coherence Grid Edition, or Oracle WebLogic Application Grid. For more information on Oracle Coherence, see <http://www.oracle.com/technology/products/coherence/index.html>.

- Third-party caches: you can create a plug-in to allow Oracle CEP to work with other, third-party cache implementations.

A cache can be considered a stage in the event processing network in which an external element (the cache) consumes or produces events; this is similar to an adapter that uses a JMS destination. A cache, however, does not have to be an actual stage in the network; another component or Spring bean can access a cache programmatically using the caching APIs.

A caching system is always configured at the application level. Other Oracle CEP applications in separate bundles can also use the caching system.

For each caching system in your application, you must create a `caching-system` element in a component configuration file. In this `caching-system` element you specify the caching system name and one or more `cache` elements. In the `cache` elements, you define optional cache configuration such as:

- Maximum cache size
- Eviction policy
- Time-to-live
- Idle time
- Write policy (write-none, write-through, or write-behind)
- Work manager

The component configuration file `caching-system` element's name element must match the EPN assembly file `caching-system` element's `id` attribute. For example, given the EPN assembly file `caching-system` element shown in [Example 12-1](#), the corresponding component configuration file `caching-system` element is shown in [Example 12-2](#).

Example 12-1 EPN Assembly File Caching System Id: `cacheSystem`

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

<wlevs:cache id="cache1">
  <wlevs:caching-system ref="cacheSystem"/>
</wlevs:cache>
```

Example 12-2 Component Configuration File Caching System Name: `cacheSystem`

```
<caching-system>
  <name>cacheSystem</name>
  <cache>
    <name>cache1</name>
    ...
  </cache>
```

```
</caching-system>
```

Similarly, the component configuration file `cache` element's `name` element must match the EPN assembly file `cache` element's `id` attribute. For example, given the EPN assembly file `cache` element shown in [Example 12-1](#), the corresponding component configuration file `cache` element is shown in [Example 12-2](#).

Example 12-3 EPN Assembly File Caching Id: cache1

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

<wlevs:cache id="cache1">
  <wlevs:caching-system ref="cacheSystem"/>
</wlevs:cache>
```

Example 12-4 Component Configuration File Caching Name: cache1

```
<caching-system>
  <name>cacheSystem</name>
  <cache>
    <name>cache1</name>
    ...
  </cache>
</caching-system>
```

You can create a `caching-system` element in any of the following component configuration files:

- The default Oracle CEP application configuration file (by default, `META-INF/wlevs/config.xml`).
- A separate configuration file.

If your application has more than one caching system, you can create a `caching-system` element for each of them in the default `config.xml` file, you can create separate XML files in `META-INF/wlevs` for each, or create a single XML file in `META-INF/wlevs` that contains the configuration for all caching systems, or even all components of your application (adapters, processors, and channels). Choose the method that best suits your development environment.

By default, Oracle CEP IDE for Eclipse creates one component configuration file and one EPN assembly file. When you add a cache to the EPN using Oracle CEP IDE for Eclipse, it adds a `cache` element to the EPN assembly file. You must manually add `caching-system` elements to the EPN assembly file and component configuration file.

Component configuration files are deployed as part of the Oracle CEP application bundle. You can later update this configuration at runtime using Oracle CEP Visualizer, the `wlevs.Admin` utility, or manipulating the appropriate JMX Mbeans directly.

For more information, see:

- [Section 1.1.5, "Component Configuration Files"](#)
- [Section 1.1.4, "EPN Assembly File"](#)
- [Section 4.3, "Creating EPN Assembly Files"](#)
- *Oracle Complex Event Processing Visualizer User's Guide*

- "wlevs.Admin Command-Line Reference" in the *Oracle Complex Event Processing Administrator's Guide*
- "Configuring JMX for Oracle CEP" in the *Oracle Complex Event Processing Administrator's Guide*

For more information on configuring caching, see:

- [Section 12.1.1, "Caching Use Cases"](#)
- [Section 12.1.2, "Additional Caching Features"](#)
- [Section 12.1.3, "Caching APIs"](#)

12.1.1 Caching Use Cases

The following sections describe the principal Oracle CEP caching use cases.

12.1.1.1 Use Case: Publishing Events to a Cache

An example of this use case is a financial application that publishes events to a cache while the financial market is open and then processes data in the cache after the market closes.

Publishing events to a cache makes them highly available or available to other Oracle CEP applications running in the server. Publishing events to a cache also allows for asynchronous writes to a secondary storage by the cache implementation. You can configure any stage in an Oracle CEP application that generates events (input adapter, channel, business POJO, or processor) to publish its events to the cache.

12.1.1.2 Use Case: Consuming Data From a Cache

Oracle CEP applications may sometimes need to access non-streaming data in order to do its work; caching this data can increase the performance of the application.

The standard components of an Oracle CEP application that are allowed direct programming access to a cache are input- and output-adapters and business POJOs.

Additionally, applications can access a cache from Oracle CQL or EPL, either by a user-defined function or directly from an Oracle CQL or EPL statement.

In the case of a user-defined function, programmers use Spring to inject the cache resource into the implementation of the function. For more information, see [Section 1.4, "Configuring Oracle CEP 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 stream data source to a processor so that querying a cache is very similar to querying a channel except that data is pulled from a cache.

An example of using Oracle CQL to query a cache is from a financial application that publishes orders and the trades used to execute the orders to a cache. At the end of the day when the markets are closed, the application queries the cache in order to find all the trades related to a particular order.

12.1.1.3 Use Case: Updating and Deleting Data in a Cache

An Oracle CEP application can both update and delete data in a cache when required.

For example, a financial application may need to update an order in the cache each time individual trades that fulfill the order are executed, or an order may need to be deleted if it has been cancelled. The components of an application that are allowed to consume data from a cache are also allowed to update it.

12.1.1.4 Use Case: Using a Cache in a Multi-Server Domain

If you build an Oracle CEP application that uses a cache, and you plan to deploy that application in a multi-server domain, then you must use a caching-system that supports a distributed cache.

In this case, you must use either Oracle Coherence or a third-party caching system that supports a distributed cache.

For more information, see:

- "Administrating Multi-Server Domains With Oracle CEP Native Clustering" in the *Oracle Complex Event Processing Administrator's Guide*
- [Section 12.3, "Configuring an Oracle Coherence Caching System and Cache"](#)
- [Section 12.4, "Configuring a Third-Party Caching System and Cache"](#)

12.1.2 Additional Caching Features

In addition to the major caching features described in the preceding sections, Oracle CEP caching includes the following features:

- Pre-load a cache with data before an application is deployed.
- Periodically refresh, invalidate, and flush the data in a cache. All these tasks happen incrementally and without halting the application or causing latency spikes.
- Dynamically update a cache's configuration.

12.1.3 Caching APIs

Oracle CEP provides a number of caching APIs that you can use in your application to perform certain tasks. The APIs are in two packages:

- `com.bea.cache.jcache`—Includes the APIs used to access a cache and create cache loader, listeners, and stores.
- `com.bea.wlevs.cache.spi`—Includes the API used to access a caching system.

The creation, configuration, and wiring of the caching systems and caches is all done using the EPN assembly file and component configuration files. This means that you typically never explicitly use the `Cache` and `CachingSystem` interfaces in your application; the only reason to use them is if you have additional requirements than the standard configuration. For example: if you want to provide integration with a third-party cache provider, then you must use the `CachingSystem` interface; if you want to perform operations on a cache that are not part of the `java.util.Map` interface, then you can use the `Cache` interface.

If you create cache listeners, loaders, or stores for an Oracle CEP local cache, then the Spring beans you write must implement the `CacheListener`, `CacheLoader`, or `CacheStore` interfaces.

If you create cache listeners, loaders, or stores for an Oracle Coherence cache, then the Spring beans you write must implement the appropriate Oracle Coherence interfaces.

If you create cache listeners, loaders, or stores for a third-party cache, then the Spring beans you write must implement the appropriate third-party cache interfaces.

For more information, see:

- [Section 12.2, "Configuring an Oracle CEP Local Caching System and Cache"](#)

- [Section 12.3, "Configuring an Oracle Coherence Caching System and Cache"](#)
- [Section 12.4, "Configuring a Third-Party Caching System and Cache"](#)
- *Oracle Fusion Middleware Java API Reference for Oracle Complex Event Processing.*

12.2 Configuring an Oracle CEP Local Caching System and Cache

You can configure your application to use the Oracle CEP local caching system and cache. The Oracle CEP local caching system is appropriate if you do not plan to deploy your application to a multi-server domain. If you plan to deploy your application to a multi-server domain, consider using an Oracle Coherence cache (see [Section 12.3, "Configuring an Oracle Coherence Caching System and Cache"](#)).

Note: It is assumed in this section that you have already created an Oracle CEP application along with its EPN assembly file and that you want to update the application to use caching. If you have not, refer to [Chapter 1, "Overview of Creating Oracle CEP Applications"](#) for details.

To configure an Oracle CEP local caching system and cache:

1. Declare the caching system in the EPN assembly file.

To declare a caching system that uses the Oracle CEP implementation declaratively in the EPN assembly file, use the `wlevs:caching-system` element without any additional attributes, as shown in the following example:

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

The value of the `id` attribute must match the name specified for the caching system in the external configuration metadata.

To allow other applications to use the caching system, specify that the caching system be advertised using the `advertise` attribute (by default set to `false`):

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

2. Declare one or more caches for this caching system in the EPN assembly file.

After you have declared a caching system for an application, you configure one or more caches using the `wlevs:cache` element:

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

The `name` attribute is optional; specify it only if the name of the cache in the caching system is different from its ID. The `wlevs:caching-system` child element references the already-declared caching system that contains the cache. You must specify this child element only if the caching system is ambiguous: there is more than one caching system declared (either implicitly or explicitly) or if the caching system is in a different application or bundle.

You can export both the caching system and the cache as an OSGI service using the `advertise` attribute.

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

```

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

```

If the cache is advertised, then a component in the EPN of an application in a *separate* bundle can then reference it. The following example shows how a processor in one bundle can use as a cache source the cache with ID `cache-id` located in a separate bundle (called `cacheprovider`):

```

<wlevs:processor id="myProcessor2">
  <wlevs:cache-source ref="cacheprovider:cache-id"/>
</wlevs:processor>

```

The caching system is responsible for creating the cache associated with a particular name and returning a reference to the cache. The resulting cache bean implements the `java.util.Map` interface.

3. Open your application's component configuration XML file using your favorite XML editor.
4. Update the `config` root element to add a `caching-system` child element; use the name child element to uniquely identify it.

This name must match the `wlevs:caching-system` element `id` attribute you specified in the EPN assembly file in step 1. This is how Oracle CEP knows to which particular caching system in the EPN assembly file this caching configuration applies.

For example, assume your configuration file already contains a processor and an adapter (contents removed for simplicity); then the updated file might look like the following

```

<?xml version="1.0" encoding="UTF-8"?>
<n1:config xmlns:n1="http://www.bea.com/ns/wlevs/config/application"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <processor>
  ...
  </processor>
  <adapter>
  ...
  </adapter>
  <caching-system>
    <name>caching-system-id</name>
  </caching-system>
</n1:config>

```

5. For each cache you want to create, update the `caching-system` element to add a cache child element; use the name child element to uniquely identify it.

This name must match the `wlevs:cache` element `id` attribute you specified in the EPN assembly file in step 2. This is how Oracle CEP knows to which particular cache in the EPN assembly file this configuration applies.

The following example shows two caches in the caching system:

```

<caching-system>
  <name>caching-system-id</name>
  <cache>
    <name>cache-id</name>
    ...
  </cache>

```

```

    <cache>
      <name>second-cache-id</name>
      ...
    </cache>
  </caching-system>

```

6. For each cache, optionally add the following elements that take simple data types to configure the cache:
- `max-size`: The number of cache elements in memory after which eviction/paging occurs. The maximum cache size is $2^{31}-1$ entries; default is 64.
 - `eviction-policy`: The eviction policy to use when `max-size` is reached. Supported values are: FIFO, LRU, LFU, and NRU; default value is LFU.
 - `time-to-live`: The maximum amount of time, in milliseconds, that an entry is cached. Default value is infinite.
 - `idle-time`: Amount of time, in milliseconds, after which cached entries are actively removed from the cache. Default value is infinite.
 - `work-manager-name`: The work manager to be used for all asynchronous operations. The value of this element corresponds to the name child element of the `work-manager` element in the server's `config.xml` configuration file.

For more information, see [Section F.44, "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>

```

7. Optionally add *either* `write-through` or `write-behind` as a child element of `cache` to specify synchronous or asynchronous writes to the cache store, respectively. By default, writes to the store are synchronous (`<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 F.44, "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:

```
<キャッシングシステム>
  <名前>キャッシングシステム-id</名前>
  <キャッシュ>
    <名前>キャッシュ-id</名前>
    <最大サイズ>100000</最大サイズ>
    <エビクションポリシー>LRU</エビクションポリシー>
    <タイム・トゥ・ライブ>3600</タイム・トゥ・ライブ>
    <write-behind>
      <buffer-size>200</buffer-size>
      <buffer-write-attempts>2</buffer-write-attempts>
      <buffer-write-timeout>200</buffer-write-timeout>
    </write-behind>
  </キャッシュ>
</キャッシングシステム>
```

8. Optionally add a cache element `listeners` child element to configure the behavior of components that listen to the cache.

Use the `asynchronous` Boolean attribute to specify whether listeners should be invoked:

- `asynchronously`: `true`.
- `synchronously`: `false`, which means listeners are invoked synchronously (Default).

The `listeners` element has a single child element, `work-manager-name`, that specifies the work manager to be used for asynchronously invoking listeners. This value is ignored if synchronous invocations are enabled. If a work manager is specified for the cache itself, this value overrides it for invoking listeners only. The value of this element corresponds to the `work-manager` element name child element in the Oracle CEP server `config.xml` configuration file.

For example:

```
<キャッシングシステム>
  <名前>キャッシングシステム-id</名前>
  <キャッシュ>
    <名前>キャッシュ-id</名前>
    <最大サイズ>100000</最大サイズ>
    <エビクションポリシー>LRU</エビクションポリシー>
    <タイム・トゥ・ライブ>3600</タイム・トゥ・ライブ>
    <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>キャッシングWM</work-manager-name>
    </listeners>
  </キャッシュ>
</キャッシングシステム>
```

```

        </listeners>
    </cache>
</caching-system>

```

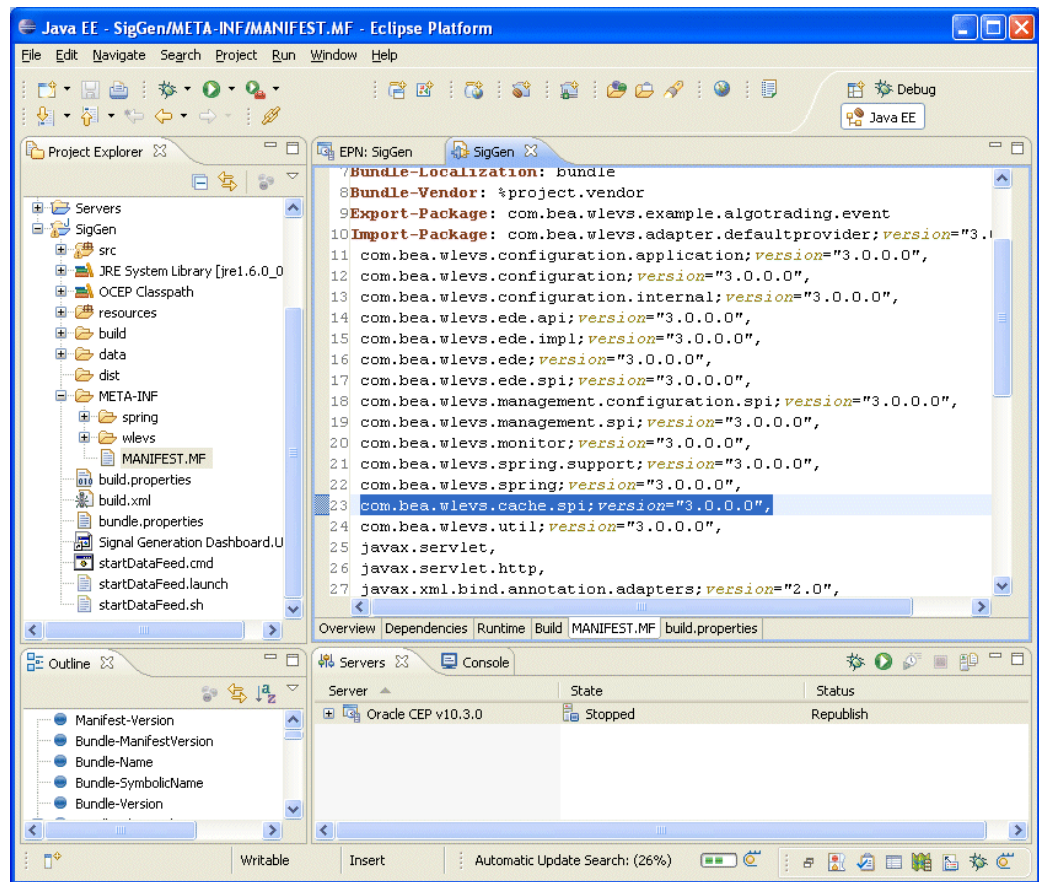
For more information, see [Section F.44, "work-manager"](#)

9. Optionally, override the default cache configuration by updating the EPN assembly file with one or more additional `cache` element child elements.
 - Specify that a cache is an event sink by configuring it as a listener to another component in the event processing network.
See [Section 12.2.1, "Configuring an Oracle CEP Local Cache as an Event Listener."](#)
 - Specify that a cache is an event source to which another component in the event processing network listens.
See [Section 12.2.2, "Configuring an Oracle CEP Local Cache as an Event Source."](#)
 - Configure a cache loader for a cache.
See [Section 12.2.3, "Configuring an Oracle CEP Local Cache Loader."](#)
 - Configure a cache store for a cache.
See [Section 12.2.4, "Configuring an Oracle CEP Local Cache Store."](#)
10. Access the Oracle CEP local cache:
 - Optionally reference the Oracle CEP local cache in a query statement.
See:
 - * [Section 12.5, "Accessing a Cache From an Oracle CQL Statement"](#)
 - * [Section 12.6, "Accessing a Cache From an EPL Statement"](#)
 - Optionally configure and program a custom adapter, business POJO, or Oracle CQL or EPL user-defined function to access the Oracle CEP local cache.
The configuration is done in the EPN assembly file and the programming is done in the Java file that implements the adapter, POJO, or user-defined function.
See:
 - * [Section 12.7, "Accessing a Cache From an Adapter"](#)
 - * [Section 12.8, "Accessing a Cache From a Business POJO"](#)
 - * [Section 12.9, "Accessing a Cache From an Oracle CQL User-Defined Function"](#)
 - * [Section 12.10, "Accessing a Cache From an EPL User-Defined Function"](#)
 - Optionally, access the Oracle CEP local cache using JMX.
See [Section 12.11, "Accessing a Cache Using JMX"](#).
11. When you assemble your application, verify that the `META-INF/MANIFEST.MF` file includes the following import as [Figure 12-1](#) shows:


```
com.bea.wlevs.cache.spi; version = "11.1.0.0"
```

If the `MANIFEST.MF` file does not include this import, update the `MANIFEST.MF` file to add this import before deploying your application.

Figure 12–1 Editing the MANIFEST.MF File



12.2.1 Configuring an Oracle CEP Local Cache as an Event Listener

An Oracle CEP local cache can be configured as an explicit listener in the event processing network in order to receive events.

For example, to specify that a cache listens to a channel, specify the `wlevs:listener` element with a reference to the cache as a child of the `wlevs:channel` element as shown below:

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

As the channel sends new events to the cache, they are inserted into the cache. If a *remove event* (an old event that exits the output window) is sent by the channel, then the event is removed from the cache.

12.2.1.1 Specifying the Key Used to Index an Oracle CEP Local Cache

When you configure an Oracle CEP local cache to be a listener, events are inserted into the cache. This section describes the variety of options available to you to specify the key used to index a cache in this instance.

If you do not explicitly specify a key, the event object itself serves as both the key and value when the event is inserted into the cache. In this case, the event class must include a valid implementation of the `equals` and `hashCode` methods that take into account the values of the key properties.

See the following for ways to explicitly specify a key:

- [Section 12.3.2.1.1, "Specifying a Key Property in EPN Assembly File"](#)
- [Section 12.3.2.1.2, "Using a Metadata Annotation to Specify a Key"](#)
- [Section 12.3.2.1.3, "Specifying a Composite Key"](#)

12.2.1.1.1 Specifying a Key Property in EPN Assembly File The first option is to specify a property name for the key property when a cache is declared in the EPN assembly file using the `key-properties` attribute, as shown in the following example:

```
<wlevs:cache id="cache-id" key-properties="key-property-name">
  <wlevs:caching-system ref="caching-system-id"/>
</wlevs:cache>
```

In this case, all events that are inserted into the cache are required to have a property of this name at runtime, otherwise Oracle CEP throws an exception.

For example, assume the event type being inserted into the cache looks something like the following; note the key property (only relevant Java source shown):

```
public class MyEvent {
    private String key;
    public MyEvent() {
    }
    public MyEvent(String key) {
        this.key = key;
    }
    public String getKey() {
        return key;
    }
    public void setKey(String key) {
        this.key = key;
    }
    ...
}
```

The corresponding declaration in the EPN assembly file would look like the following:

```
<wlevs:cache id="cache-id" key-properties="key">
  <wlevs:caching-system ref="caching-system-id"/>
</wlevs:cache>
```

12.2.1.1.2 Using a Metadata Annotation to Specify a Key The second option is to use the metadata annotation `com.bea.wlevs.ede.api.Key` to annotate the event property in the Java class that implements the event type. This annotation does not have any attributes.

To use a metadata annotation to specify a key:

1. Import the `com.bea.wlevs.ede.api.Key` package.

For more information, see [Section 4.7.5, "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;
    }
    ...
}

```

12.2.1.1.3 Specifying a Composite Key The final option is to use the `key-class` attribute of the `wlevs:cache` element to specify a composite key in which multiple properties form the key. The value of the `key-class` attribute must be a JavaBean whose public fields match the fields of the event class. The matching is done according to the field name. For example:

```

<wlevs:cache id="cache-id" key-class="key-class-name">
  <wlevs:caching-system ref="caching-system-id"/>
</wlevs:cache>

```

12.2.2 Configuring an Oracle CEP Local Cache as an Event Source

An Oracle CEP local cache can be configured as a source of events to which another component in the event processing network listens. The listening component can be an adapter or a standard Spring bean. Any component that listens to a cache must implement the `com.bea.cache.jcache.CacheListener` interface. The following example shows how to configure a cache to be an event source for a Spring bean:

```

<wlevs:caching-system id="caching-system-id"/>
...
<wlevs:cache id="cache-id" name="alternative-cache-name">
  <wlevs:caching-system ref="caching-system-id"/>
  <wlevs:cache-listener ref="cache-listener-id" />
</wlevs:cache>
...
<bean id="cache-listener-id" class="wlevs.example.MyCacheListener"/>

```

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

12.2.3 Configuring an Oracle CEP Local Cache Loader

An Oracle CEP local cache loader is an object that loads objects into an Oracle CEP local cache. You configure a cache loader by using the `wlevs:cache-loader` child element of the `wlevs:cache` element to specify the bean that does the loading work, as shown in the following example:

```

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

```

```

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

```

In the example, the `cache-loader-id` Spring bean does the cache loading work; it is referenced by the cache using the `ref` attribute of the `wlevs:cache-loader` child element.

You must program the `wlevs.example.MyCacheLoader` class yourself, and it must implement the `com.bea.cache.jcache.CacheLoader` interface. This interface includes the `load` method to customize the loading of a single object into the cache; Oracle CEP calls this method when the requested object is not in the cache. The interface also includes `loadAll` methods that you implement to customize the loading of the entire cache.

12.2.4 Configuring an Oracle CEP Local Cache Store

You can configure an Oracle CEP local cache with a custom store that is responsible for writing data from the cache to a backing store, such as a table in a database. You configure a cache store by using the `wlevs:cache-store` child element of the `wlevs:cache` element to specify the bean that does the actual storing work, as shown in the following example:

```

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

```

In the example, the `cache-store-id` Spring bean does the work of writing the data from the cache to the backing store; it is referenced by the cache using the `ref` attribute of the `wlevs:cache-store` child element.

You must program the `wlevs.example.MyCacheStore` class yourself, and it must implement the `com.bea.cache.jcache.CacheStore` interface. This interface includes the `store` method that stores the data in the backing store using the passed key; Oracle CEP calls this method when it inserts data into the cache. The interface also includes the `storeAll` method for storing a batch of data to a backing store in the case that you have configured asynchronous writes for a cache with the `write-behind` configuration element.

12.3 Configuring an Oracle Coherence Caching System and Cache

You can configure your application to use the Oracle Coherence caching system and cache. Use this caching system if you plan to deploy your application to a multi-server domain.

Using Oracle Coherence, only the first caching-system can be configured in a server. Oracle CEP will ignore any other caching systems you might configure.

Note: Before you can legally use Oracle CEP with Oracle Coherence, you must obtain a valid Coherence license such as a license for Coherence Enterprise Edition, Coherence Grid Edition, or Oracle WebLogic Application Grid. For more information on Oracle Coherence, see <http://www.oracle.com/technology/products/coherence/index.html>.

Note: It is assumed in this section that you have already created an Oracle CEP application along with its EPN assembly file and that you want to update the application to use caching. If you have not, refer to [Chapter 1, "Overview of Creating Oracle CEP Applications"](#) for details.

To configure an Oracle Coherence caching system and cache:

1. 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 12.3.5, "Configuring a Shared Oracle Coherence Cache"](#).

2. Declare the Oracle Coherence caching system in the EPN assembly file.

To declare a caching system that uses the Oracle Coherence implementation declaratively in the EPN assembly file, use the `wlevs:caching-system` element as shown in the following example:

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

The value of the `id` attribute must match the name you specify for the caching system in the application's component configuration file (see step 4).

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="coherence"
advertise="false"/>
...
<wlevs:cache id="myCache" advertise="false">
  <wlevs:caching-system ref="caching-system-id"/>
</wlevs:cache>
```

The `wlevs:cache` element `id` attribute is mandatory. This attribute maps to the name of a cache in the Oracle Coherence configuration files (see step 4).

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.

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.

For more information, see [Section 12.3.5, "Configuring a Shared Oracle Coherence Cache"](#).

4. Configure the caching system and its caches by updating the caching configuration file for the application.
See [Section 12.3.1, "Configuring the Oracle Coherence Caching System and Caches."](#)
5. Optionally, override the default cache configuration by updating the EPN assembly file with one or more additional `cache` element child elements.
 - Specify that a cache is an event sink by configuring it as a listener to another component in the event processing network.
See [Section 12.3.1, "Configuring the Oracle Coherence Caching System and Caches."](#)
 - Specify that a cache is an event source to which another component in the event processing network listens.
See [Section 12.3.2, "Configuring an Oracle Coherence Cache as an Event Listener."](#)
 - Configure *either* a `<wlevs:cache-loader>` or a `<wlevs:cache-store>` child element of the `<wlevs:cache>` element in the EPN assembly file, but not both.

This is because Oracle Coherence combines the loader and store into a single component. You specify a cache loader when the backing store is read-only and a cache store when the backing store is read-write
See [Section 12.3.4, "Configuring an Oracle Coherence Cache Loader or Store."](#)
6. Access the Oracle Coherence cache:
 - Optionally reference the Oracle Coherence cache in a query statement.
See:
 - * [Section 12.5, "Accessing a Cache From an Oracle CQL Statement"](#)
 - * [Section 12.6, "Accessing a Cache From an EPL Statement"](#)
 - Optionally configure and program a custom adapter, business POJO, or Oracle CQL or EPL user-defined function to access the Oracle Coherence cache.

The configuration is done in the EPN assembly file and the programming is done in the Java file that implements the adapter, POJO, or user-defined function.
See:
 - * [Section 12.7, "Accessing a Cache From an Adapter"](#)
 - * [Section 12.8, "Accessing a Cache From a Business POJO"](#)
 - * [Section 12.9, "Accessing a Cache From an Oracle CQL User-Defined Function"](#)
 - * [Section 12.10, "Accessing a Cache From an EPL User-Defined Function"](#)
 - Optionally, access the Oracle Coherence cache using JMX.

See [Section 12.11, "Accessing a Cache Using JMX"](#).

7. Edit your `MANIFEST.MF` to import package `com.bea.wlevs.cache.spi`.

For more information, see [Section 4.7.5, "How to Import a Package"](#).

8. Assemble and deploy your application.

For more information, see [Chapter 24, "Assembling and Deploying Oracle CEP Applications"](#).

12.3.1 Configuring the Oracle Coherence Caching System and Caches

Oracle CEP leverages the native configuration provided by Oracle Coherence. You do this by packaging the following two Oracle Coherence configuration files, with the indicated names, in the application bundle that uses the Oracle Coherence cache:

- `coherence-cache-config.xml`—Oracle Coherence cache configuration information. Individual caches are identified with the `cache-name` element; the value of this element maps to the `id` attribute of the `wlevs:cache` element in the EPN assembly file. See [Section 12.3.1.1, "The coherence-cache-config.xml File"](#) for information about this file as well as an example of the mapping.
- `tangosol-coherence-override.xml`—Oracle Coherence cluster configuration. See [Section 12.3.1.2, "The tangosol-coherence-override.xml File"](#) for information about this file as well as an example.

When assembling your application, consider the following:

- `coherence-cache-config.xml` is a per-application configuration file; put this file in the `META-INF/wlevs/coherence` directory of the bundle JAR. Note that this directory is different from the directory that stores the component configuration file for the local in-memory Oracle CEP caching provider (`META-INF/wlevs`).
- `tangosol-coherence-override.xml` is a global per-server file (referred to as "operational configuration" in the Oracle Coherence documentation); put this file in the Oracle CEP server `config` directory.

Update your application configuration file as [Example 12-5](#) shows:

Example 12-5 Application Configuration File: Coherence Cache

```
<coherence-caching-system>
  <name>caching-system-id</name>
  <coherence-cache-config>
    ../wlevs/coherence/coherence-cache-config.xml
  </coherence-cache-config>
</coherence-caching-system>
```

When you declare that a caching system uses the Oracle Coherence provider, be sure that all of the caches of this caching system also map to an Oracle Coherence configuration and not an Oracle CEP local configuration, or Oracle CEP throws an exception.

12.3.1.1 The coherence-cache-config.xml File

The `coherence-cache-config.xml` file is the basic Oracle Coherence configuration file and must conform to the Oracle Coherence DTDs, as is true for any Oracle Coherence application.

The following sample shows a simple configuration. See the explanation after the sample for information about the sections in bold.

```
<?xml version="1.0"?>
<!DOCTYPE cache-config SYSTEM "cache-config.dtd">
<cache-config>
  <caching-scheme-mapping>
    <cache-mapping>
      <cache-name>myCoherenceCache</cache-name>
      <scheme-name>new-replicated</scheme-name>
    </cache-mapping>
    <cache-mapping>
      <cache-name>myLoaderCache</cache-name>
      <scheme-name>test-loader-scheme</scheme-name>
    </cache-mapping>
    <cache-mapping>
      <cache-name>myStoreCache</cache-name>
      <scheme-name>test-store-scheme</scheme-name>
    </cache-mapping>
  </caching-scheme-mapping>
  <caching-schemes>
    <replicated-scheme>
      <scheme-name>new-replicated</scheme-name>
      <service-name>ReplicatedCache</service-name>
      <backing-map-scheme>
        <local-scheme>
          <scheme-ref>my-local-scheme</scheme-ref>
        </local-scheme>
      </backing-map-scheme>
    </replicated-scheme>
    <local-scheme>
      <scheme-name>my-local-scheme</scheme-name>
      <eviction-policy>LRU</eviction-policy>
      <high-units>100</high-units>
      <low-units>50</low-units>
    </local-scheme>
    <local-scheme>
      <scheme-name>test-loader-scheme</scheme-name>
      <eviction-policy>LRU</eviction-policy>
      <high-units>100</high-units>
      <low-units>50</low-units>
      <cachestore-scheme>
      <class-scheme>
      <class-factory-name>
      com.bea.wlevs.cache.coherence.configuration.SpringFactory
      </class-factory-name>
      <method-name>getLoader</method-name>
      <init-params>
      <init-param>
      <param-type>java.lang.String</param-type>
      <param-value>{cache-name}</param-value>
      </init-param>
      </init-params>
      </class-scheme>
      </cachestore-scheme>
    </local-scheme>
    <local-scheme>
      <scheme-name>test-store-scheme</scheme-name>
      <eviction-policy>LRU</eviction-policy>
      <high-units>100</high-units>
      <low-units>50</low-units>
      <cachestore-scheme>
      <class-scheme>
      <class-factory-name>
      com.bea.wlevs.cache.coherence.configuration.SpringFactory
      </class-factory-name>
    </local-scheme>
  </caching-schemes>
</cache-config>
```

```

    <method-name>getStore</method-name>
    <init-params>
      <init-param>
        <param-type>java.lang.String</param-type>
        <param-value>{cache-name}</param-value>
      </init-param>
    </init-params>
  </class-scheme>
</cachestore-scheme>
</local-scheme>
</caching-schemes>
</cache-config>

```

In the Oracle Coherence configuration file, the `cache-name` element that is a child element of `cache-mapping` identifies the name of the Oracle Coherence cache. The value of this element must exactly match the value of the `id` attribute of the `wlevs:cache` element in the EPN assembly file. For example, the following EPN assembly file snippet refers to the `myCoherenceCache` cache in the Oracle Coherence configuration file:

```

<wlevs:cache id="myCoherenceCache" advertise="false">
  <wlevs:caching-system ref="coherence-cache"/>
  <wlevs:cache-loader ref="localLoader"/>
  <wlevs:cache-listener ref="localListener"/>
</wlevs:cache>

```

The Oracle Coherence configuration file illustrates another requirement when using Oracle Coherence with Oracle CEP: an Oracle Coherence factory must be declared when using Spring to configure a loader or store for a cache. You do this using the `cachestore-scheme` element in the Oracle Coherence configuration file to specify a factory class that allows Oracle Coherence to call into Oracle CEP and retrieve a reference to the loader or store that is configured for the cache. The only difference between configuring a loader or store is that the `method-name` element has a value of `getLoader` when a loader is used and `getStore` when a store is being used. You pass the cache name to the factory as an input parameter.

Refer to your Oracle Coherence documentation (see <http://www.oracle.com/technology/products/coherence/index.html>) for detailed information about the `coherence-cache-config.xml` file.

12.3.1.2 The `tangosol-coherence-override.xml` File

The `tangosol-coherence-override.xml` file configures Oracle Coherence caching. You may 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 CEP starts up; this can cause problems and sometimes even prevent Oracle CEP from starting.

Refer to your Oracle Coherence documentation (see <http://www.oracle.com/technology/products/coherence/index.html>) for detailed information about the `tangosol-coherence-override.xml` file.

For more information on Oracle CEP clusters, see "Administrating Multi-Server Domains With Oracle CEP Native Clustering" in the *Oracle Complex Event Processing Administrator's Guide*.

12.3.2 Configuring an Oracle Coherence Cache as an Event Listener

An Oracle Coherence cache can be configured as an explicit listener in the event processing network in order to receive events.

For example, to specify that a cache listens to a channel, specify the `wlevs:listener` element with a reference to the cache as a child of the `wlevs:channel` element as shown below:

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

As the channel sends new events to the cache, they are inserted into the cache. If a *remove event* (an old event that exits the output window) is sent by the channel, then the event is removed from the cache.

12.3.2.1 Specifying the Key Used to Index an Oracle Coherence Cache

When you configure an Oracle Coherence cache to be a listener, events are inserted into the cache. This section describes the variety of options available to you to specify the key used to index a cache in this instance.

If you do not explicitly specify a key, the event object itself serves as both the key and value when the event is inserted into the cache. In this case, the event class must include a valid implementation of the `equals` and `hashCode` methods that take into account the values of the key properties.

See the following for ways to explicitly specify a key:

- [Section 12.3.2.1.1, "Specifying a Key Property in EPN Assembly File"](#)
- [Section 12.3.2.1.2, "Using a Metadata Annotation to Specify a Key"](#)
- [Section 12.3.2.1.3, "Specifying a Composite Key"](#)

12.3.2.1.1 Specifying a Key Property in EPN Assembly File The first option is to specify a property name for the key property when a cache is declared in the EPN assembly file using the `key-properties` attribute, as shown in the following example:

```
<wlevs:cache id="myCache" key-properties="key-property-name">
  <wlevs:caching-system ref="caching-system-id"/>
</wlevs:cache>
```

In this case, all events that are inserted into the cache are required to have a property of this name at runtime, otherwise Oracle CEP throws an exception.

For example, assume the event type being inserted into the cache looks something like the following; note the key property (only relevant Java source shown):

```

public class MyEvent {
    private String key;
    public MyEvent() {
    }
    public MyEvent(String key) {
        this.key = key;
    }
    public String getKey() {
        return key;
    }
    public void setKey(String key) {
        this.key = key;
    }
    ...
}

```

The corresponding declaration in the EPN assembly file would look like the following:

```

<wlevs:cache id="myCache" key-properties="key">
    <wlevs:caching-system ref="caching-system-id"/>
</wlevs:cache>

```

12.3.2.1.2 Using a Metadata Annotation to Specify a Key The second option is to use the metadata annotation `com.bea.wlevs.ede.api.Key` to annotate the event property in the Java class that implements the event type. This annotation does not have any attributes.

To use a metadata annotation to specify a key:

1. Import the `com.bea.wlevs.ede.api.Key` package.

For more information, see [Section 4.7.5, "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;
    }
    ...
}

```

12.3.2.1.3 Specifying a Composite Key The final option is to use the `key-class` attribute of the `wlevs:cache` element to specify a composite key in which multiple properties form the key. The value of the `key-class` attribute must be a JavaBean whose public fields match the fields of the event class. The matching is done according to the field name. For example:

```

<wlevs:cache id="myCache" key-class="key-class-name">
    <wlevs:caching-system ref="caching-system-id"/>

```

```
</wlevs:cache>
```

12.3.3 Configuring an Oracle Coherence Cache as an Event Source

You can configure an Oracle Coherence cache as a source of events to which another component in the event processing network listens. The listening component can be an adapter or a standard Spring bean. Any component that listens to a cache must implement the `com.tangosol.util.MapListener` interface. The following example shows how to configure an Oracle Coherence cache to be an event source for a Spring bean:

```
<wlevs:coherence-caching-system id="caching-system-id"/>
...
<wlevs:cache id="myCache" advertise="false">
  <wlevs:caching-system ref="caching-system-id"/>
  <wlevs:cache-loader ref="localLoader"/>
  <wlevs:cache-listener ref="localListener"/>
</wlevs:cache>
<bean id="localListener"
      class="com.bea.wlevs.example.provider.coherence.LocalListener"/>
```

In the example, the `cache-listener-id` Spring bean listens to events coming from the cache; the class that implements this component, `com.bea.wlevs.example.provider.coherence.LocalListener`, must implement the appropriate Oracle Coherence-specific Java interfaces such as `com.tangosol.util.MapListener`. You must program this `LocalListener` class yourself as [Example 12-6](#) shows.

Example 12-6 Oracle Coherence Cache LocalListener Implementation

```
package com.bea.wlevs.example.provider.coherence;

import com.tangosol.util.MapEvent;
import com.tangosol.util.MapListener;

public class LocalListener implements MapListener {
    public static int deleted = 0;
    public static int inserted = 0;
    public static int updated = 0;

    public void entryDeleted(MapEvent event) {
        deleted++;
    }
    public void entryInserted(MapEvent event) {
        inserted++;
    }
    public void entryUpdated(MapEvent event) {
        updated++;
    }
}
```

12.3.4 Configuring an Oracle Coherence Cache Loader or Store

Using an Oracle Coherence cache, you may configure either a `wlevs:cache-loader` or a `wlevs:cache-store` child element of the `wlevs:cache` element in the EPN assembly file, but not both. This is because Oracle Coherence combines the loader and store into a single component:

- Specify a cache loader when the backing store is read-only.
See [Section 12.3.4.1, "Configuring an Oracle Coherence Cache Loader."](#)
- Specify a cache store when the backing store is read-write.

See [Section 12.3.4.2, "Configuring an Oracle Coherence Cache Store."](#)

12.3.4.1 Configuring an Oracle Coherence Cache Loader

In [Example 12-7](#), the `localLoader` Spring bean loads events into the Oracle Coherence cache when the backing store is read-only. If the backing store is read-write, use a cache store (see [Section 12.3.4.2, "Configuring an Oracle Coherence Cache Store"](#)).

The class that implements this component, `com.bea.wlevs.example.provider.coherence.LocalLoader`, must implement the appropriate Oracle Coherence-specific Java interfaces such as `com.tangosol.net.cache.CacheLoader`. You must program this `LocalLoader` class yourself as [Example 12-8](#) shows.

Example 12-7 Oracle Coherence Cache EPN Assembly File for a Cache Loader

```
<wlevs:coherence-caching-system id="caching-system-id"/>
<wlevs:cache id="myCache" advertise="false">
  <wlevs:caching-system ref="caching-system-id"/>
  <wlevs:cache-loader ref="localLoader"/>
</wlevs:cache>
<bean id="localLoader"
      class="com.bea.wlevs.example.provider.coherence.LocalLoader"/>
```

Example 12-8 Oracle Coherence Cache LocalLoader Implementation

```
package com.bea.wlevs.example.provider.coherence;

import java.util.Collection;
import java.util.HashMap;
import java.util.HashSet;
import java.util.Map;
import java.util.Set;

import com.bea.wlevs.example.provider.event.ProviderData;
import com.tangosol.net.cache.CacheLoader;

public class LocalLoader implements CacheLoader {
    public static int loadCount = 0;
    public static Set keys = new HashSet();

    public LocalLoader() {
    }
    public Object load(Object key) {
        loadCount++;
        keys.add(key);
        return new ProviderData((String) key);
    }
    public Map loadAll(Collection keys) {
        Map result = new HashMap();

        for (Object key : keys) {
            result.put(key, load(key));
        }
        return result;
    }
}
```

12.3.4.2 Configuring an Oracle Coherence Cache Store

In [Example 12–9](#), the `localStore` Spring bean loads events into the cache when the backing store is read-write. If the backing store is read-only, use a cache loader (see [Section 12.3.4.1, "Configuring an Oracle Coherence Cache Loader"](#)).

The class that implements this component, `com.bea.wlevs.example.provider.coherence.LocalStore`, must implement the appropriate Oracle Coherence-specific Java interfaces such as `com.tangosol.net.cache.CacheStore`. You must program this `LocalStore` class yourself as [Example 12–10](#) shows.

Example 12–9 Oracle Coherence Cache EPN Assembly File for a Cache Store

```
<wlevs:coherence-caching-system id="caching-system-id"/>
<wlevs:cache id="myCache" advertise="false">
  <wlevs:caching-system ref="caching-system-id"/>
  <wlevs:cache-store ref="localStore"/>
</wlevs:cache>
<bean id="localStore"
      class="com.bea.wlevs.example.provider.coherence.LocalStore"/>
```

Example 12–10 Oracle Coherence Cache LocalStore Implementation

```
package com.bea.wlevs.example.provider.coherence;

import java.util.Collection;
import java.util.HashMap;
import java.util.Map;
import java.util.Set;

import com.bea.wlevs.example.provider.event.ProviderData;
import com.tangosol.net.cache.CacheStore;

public class LocalStore implements CacheStore {
    public static int eraseCount = 0;
    public static int storeCount = 0;
    public static int loadCount = 0;

    public void erase(Object key) {
        eraseCount++;
    }
    public void eraseAll(Collection keys) {
        for (Object key : keys) {
            erase(key);
        }
    }
    public void store(Object key, Object value) {
        //
        // Do the store operation here.
        //
    }
    public void storeAll(Map entries) {
        for (Map.Entry entry : (Set <Map.Entry>)entries.entrySet()) {
            store(entry.getKey(), entry.getValue());
        }
    }
    public Object load(Object key) {
        loadCount++;
        return new ProviderData((String) key);
    }
    public Map loadAll(Collection keys) {
        Map result = new HashMap();
        for (Object key : keys) {
```



```

        result.put(key, load(key));
    }
    return result;
}
}

```

12.3.5 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 CEP server, you should never configure multiple instances of the same cache with a loader or store. You might inadvertently do this by employing multiple applications that each configure the same Oracle Coherence cache with a loader or store in their respective EPN assembly file. If you do this, Oracle CEP throws an exception.

If multiple application bundles need to share Oracle Coherence caches, then you should put the EPN assembly file that contains the appropriate `wlevs:cache` and `wlevs:caching-system` in a separate bundle and set their `advertise` attributes to `true`.

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>

```

12.4 Configuring a Third-Party Caching System and Cache

You can configure your application to use a third-party caching system and cache.

Note: It is assumed in this section that you have already created an Oracle CEP application along with its EPN assembly file and that you want to update the application to use caching. If you have not, refer to [Chapter 1, "Overview of Creating Oracle CEP Applications"](#) for details.

To configure a third-party caching system and cache:

1. Create a plug-in to define the third-party caching system as an Oracle CEP caching system provider.

This involves:

- Implementing the `com.bea.wlevs.cache.spi.CachingSystem` interface
- Creating a factory that creates caching systems of this type.
- Registering the factory with an attribute that identifies its provider type.

2. Declare the caching system in the EPN assembly file.

Use the `wlevs:caching-system` element to declare a third-party implementation; use the `class` or `provider` attributes to specify additional information.

For simplicity, you can include the third-party implementation code inside the Oracle CEP application bundle itself to avoid having to import or export packages and managing the lifecycle of a separate bundle that contains the third-party implementation. In this case the `wlevs:caching-system` element appears in the EPN assembly file as shown in the following example:

```
<wlevs:caching-system id="caching-system-id"
    class="third-party-implementation-class"/>
```

The `class` attribute specifies a Java class that must implement the `com.bea.wlevs.cache.spi.CachingSystem` interface. For details about this interface, see the *Oracle Fusion Middleware Java API Reference for Oracle Complex 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 CEP application that is using it. In this case, you must create a *separate* bundle whose Spring application context includes the `wlevs:caching-system` element, with the `advertise` attribute mandatory:

```
<wlevs:caching-system id="caching-system-id"
    class="third-party-implementation-class"
    advertise="true"/>
```

Alternatively, if you want to decouple the implementation bundle from the bundle that references it, or you are plugging in a caching implementation that supports multiple caching systems per Java process, you can specify a factory as a provider:

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

The factory class (`the.factory.class.name` in the example) must implement the `com.bea.wlevs.cache.spi.CachingSystemFactory` interface. This interface has a single method, `create`, that returns a `com.bea.wlevs.cache.spi.CachingSystem` instance.

You must deploy this bundle alongside the application bundle so that the latter can start using it.

3. Declare one or more caches for this caching system in the EPN assembly file.

After you have declared a caching system for an application, you configure one or more caches using the `wlevs:cache` element:

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

The `name` attribute is optional; specify it only if the name of the cache in the caching system is different from its ID. The `wlevs:caching-system` child element references the already-declared caching system that contains the cache. You must specify this child element only if the caching system is ambiguous: there

is more than one caching system declared (either implicitly or explicitly) or if the caching system is in a different application or bundle.

You can export both the caching system and the cache as an OSGI service using the `advertise` attribute.

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

If the cache is advertised, then a component in the EPN of an application in a *separate* bundle can then reference it. The following example shows how a processor in one bundle can use as a cache source the cache with ID `cache-id` located in a separate bundle (called `cacheprovider`):

```
<wlevs:processor id="myProcessor2">
  <wlevs:cache-source ref="cacheprovider:cache-id"/>
</wlevs:processor>
```

The caching system is responsible for creating the cache associated with a particular name and returning a reference to the cache. The resulting cache bean implements the `java.util.Map` interface.

4. Configure the third-party caching system and its caches by updating the third-party caching configuration file or files for the application.

Refer to your third-party cache documentation.

5. Optionally, override the default third-party cache configuration by updating the appropriate configuration file with one or more additional `cache` element child elements.

- Specify that a cache is an event sink by configuring it as a listener to another component in the event processing network.

Refer to your third-party cache documentation.

- Specify that a cache is an event source to which another component in the event processing network listens.

Refer to your third-party cache documentation.

- Configure a cache loader or store.

Refer to your third-party cache documentation.

6. Access the third-party cache:

- Optionally reference the third-party cache in a query statement.

See:

- * [Section 12.5, "Accessing a Cache From an Oracle CQL Statement"](#)

- * [Section 12.6, "Accessing a Cache From an EPL Statement"](#)

- Optionally configure and program a custom adapter, business POJO, or Oracle CQL or EPL user-defined function to access the third-party cache.

The configuration is done in the EPN assembly file and the programming is done in the Java file that implements the adapter, POJO, or user-defined function.

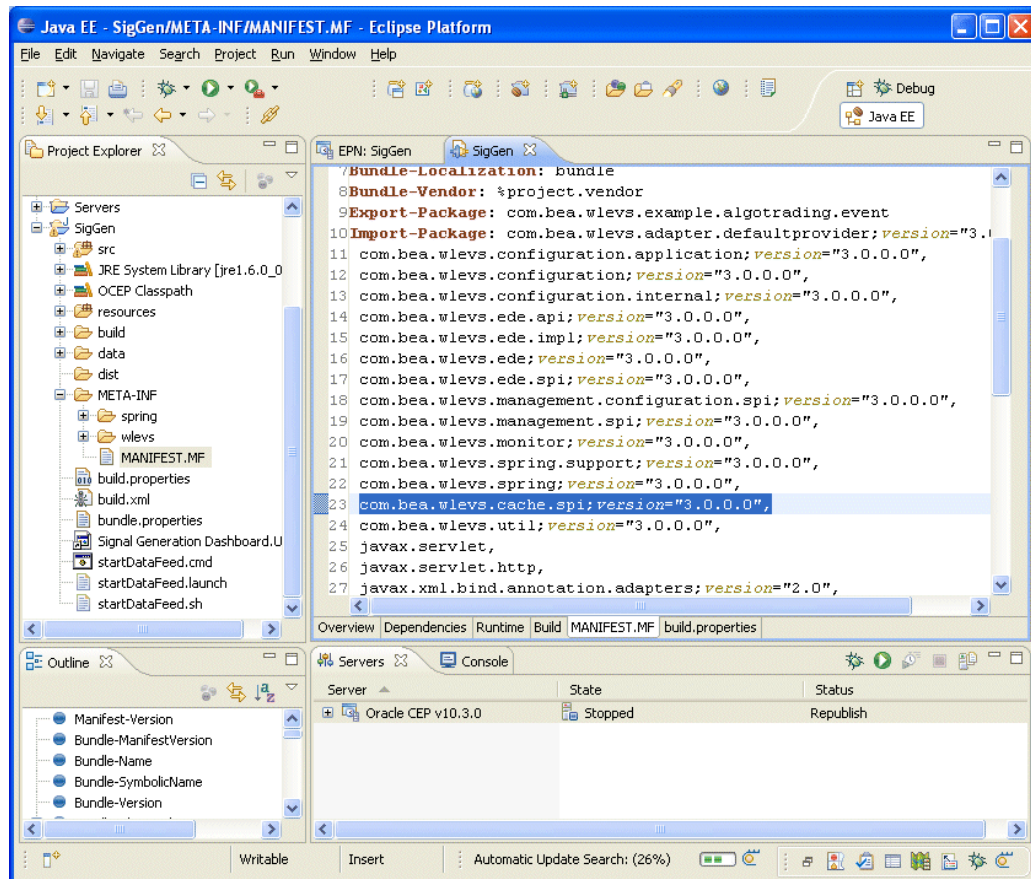
See:

- * [Section 12.7, "Accessing a Cache From an Adapter"](#)
 - * [Section 12.8, "Accessing a Cache From a Business POJO"](#)
 - * [Section 12.9, "Accessing a Cache From an Oracle CQL User-Defined Function"](#)
 - * [Section 12.10, "Accessing a Cache From an EPL User-Defined Function"](#)
- Optionally, access the third-party cache using JMX.
- See [Section 12.11, "Accessing a Cache Using JMX"](#).
7. When you assemble your application, verify that the META-INF/MANIFEST.MF file includes the following import as [Figure 12–1](#) shows:

```
com.bea.wlevs.cache.spi; version = "11.1.0.0"
```

If the MANIFEST.MF files does not include this import, update the MANIFEST.MF file to add this import before deploying your application.

Figure 12–2 *Editing the MANIFEST.MF File*



12.5 Accessing a Cache From an Oracle CQL Statement

You can reference a cache from an Oracle CQL statement in much the same way you reference a channel; this feature enables you to enrich standard streaming data with data from a separate source. [Example 12–11](#) shows a valid Oracle CQL query that joins trade events from a standard channel named S1 with stock symbol data from a cache named stockCache:

Example 12–11 Valid Oracle CQL Query Against a Cache

```
SELECT S1.symbol, S1.lastPrice, stockCache.description
FROM S1 [Now], stockCache
WHERE S1.symbol = stockCache.symbol
```

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 CEP 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 CEP *pulls* the data rather than it being *pushed*, as is the case with a channel. This means that, continuing with [Example 12–11](#), the query executes only when a channel pushes a trade event to the query; the stock symbol data in the cache never causes a query to execute, it is only pulled by the query when needed.

- You must specify all of the key properties 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, group).

The following query is invalid because although the WHERE clause specifies both key properties (id and group), the query does not supply a value for one of the key properties (id):

```
select count(*) as n from S [now], C
where S.group = C.group and S.id is not null
```

A valid query is:

```
select count(*) as n from S [now], C
where S.group = C.group and S.id = C.id
```

For instructions on specifying the cache key, see:

- [Section 12.2.1.1, "Specifying the Key Used to Index an Oracle CEP Local Cache"](#)
- [Section 12.3.2.1, "Specifying the Key Used to Index an Oracle Coherence Cache"](#)

- Joins must be executed only by referencing the cache key.
- You cannot use a cache in a view. Instead, use a join.
- Only a single channel source may occur in the FROM clause of an Oracle CQL statement that joins cache data source(s).
- If the cache is a processor source, you connect the cache directly to the processor on the EPN as [Figure 12–3](#) shows.

- If the cache is a processor sink, you connect the processor to the cache using a channel as [Figure 12–4](#) shows.

12.5.1 How to Access a Cache From an Oracle CQL Statement

This section describes how to reference a cache in an Oracle CQL query statement.

This procedure assumes that you have already configured the caching system and caches. For more information, see:

- [Section 12.2, "Configuring an Oracle CEP Local Caching System and Cache"](#)
- [Section 12.3, "Configuring an Oracle Coherence Caching System and Cache"](#)
- [Section 12.4, "Configuring a Third-Party Caching System and Cache"](#)

To access a cache from an Oracle CQL statement:

1. If you have not already done so, create the event type that corresponds to the cache data and register it in the event repository.

See [Chapter 2.1.4, "Creating Oracle CEP Event Types"](#).

2. Specify the key properties for the data in the cache.

For instructions on specifying the cache key, see:

- [Section 12.2.1.1, "Specifying the Key Used to Index an Oracle CEP Local Cache"](#)
- [Section 12.3.2.1, "Specifying the Key Used to Index an Oracle Coherence Cache"](#)

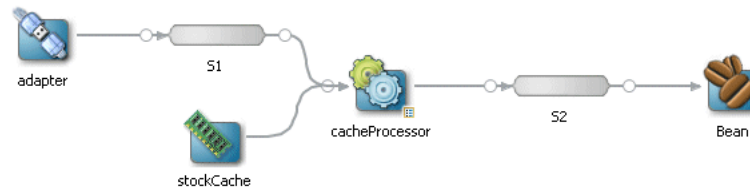
3. In the EPN assembly file, update the configuration of the cache to declare the event type of its values; use the `value-type` attribute of the `wlevs:cache` element. For example:

```
<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 12–3](#) shows.

Figure 12-3 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 12-4](#) shows.

Figure 12-4 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>
  
```

12.6 Accessing a Cache From an EPL Statement

You can reference a cache from an EPL statement in much the same way you reference a channel; this feature enables you to enrich standard streaming data with data from a separate source. For example, the following EPL query joins trade events from a standard channel with company data from a cache:

```

INSERT INTO EnrichedTradeEvent
SELECT trade.symbol, trade.price, trade.numberOfShares, company.name
FROM TradeEvent trade RETAIN 8 hours, Company company
WHERE trade.symbol = company.id
  
```


In the example, both `TradeEvent` and `Company` are event types registered in the repository, but they have been configured in such a way that `TradeEvents` come from a standard stream of events but `Company` maps to a cache in the event processing network. This configuration happens outside of the EPL query, which means that the source of the data is transparent in the query itself.

When you use data from a cache in an EPL query, Oracle CEP *pulls* the data rather than it being *pushed*, as is the case with a channel. This means that, continuing with the preceding sample, the query executes only when a channel pushes a trade event to the query; the company data in the cache never causes a query to execute, it is only pulled by the query when needed.

You must abide by these restrictions when using a cache in an EPL query:

- You must specify the key properties for data in the cache.
For instructions on specifying the cache key, see:
 - [Section 12.2.1.1, "Specifying the Key Used to Index an Oracle CEP Local Cache"](#)
 - [Section 12.3.2.1, "Specifying the Key Used to Index an Oracle Coherence Cache"](#)
- Joins must be executed only by referencing the cache key.
- You cannot specify a `RETAIN` clause for data pulled from a cache. If an event type that gets its data from a cache is included in a `RETAIN` clause, Oracle CEP ignores it.
- You cannot use a cache in a correlated sub-query. Instead, use a join.
- Only a single channel source may occur in the `FROM` clause of an EPL statement that joins cache data source(s). Using multiple cache sources and parameterized SQL queries is supported.

12.6.1 How To Access a Cache From an EPL Statement

This section describes how to reference a cache in an EPL query statement.

This procedure assumes that you have already configured the caching system and caches. For more information, see:

- [Section 12.2, "Configuring an Oracle CEP Local Caching System and Cache"](#)
- [Section 12.3, "Configuring an Oracle Coherence Caching System and Cache"](#)
- [Section 12.4, "Configuring a Third-Party Caching System and Cache"](#)

To access a cache from an EPL statement:

1. If you have not already done so, create the event type that corresponds to the cache data, such as `Company` in the preceding example, and registered it in the event repository. See [Chapter 2.1.4, "Creating Oracle CEP Event Types"](#).
2. Specify the key properties for the data in the cache. There are a variety of ways to do this; see
 - [Section 12.2.1.1, "Specifying the Key Used to Index an Oracle CEP Local Cache"](#)
 - [Section 12.3.2.1, "Specifying the Key Used to Index an Oracle Coherence Cache"](#)

3. In the EPN assembly file, update the configuration of the cache in the EPN assembly file to declare the event type of its values; use the `value-type` attribute of the `wlevs:cache` element. For example:

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

The `value-type` attribute specifies the type for the values contained in the cache. This must 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.

12.7 Accessing a Cache From an Adapter

An adapter can also be injected with a cache using the standard Spring mechanism for referencing another bean. A cache bean implements the `java.util.Map` interface which is what the adapter uses to access the injected cache.

First, the configuration of the adapter in the EPN assembly file must be updated with a `wlevs:instance-property` child element, as shown in the following example:

```
<wlevs:caching-system id="caching-system-id"/>
...
<wlevs:cache id="cache-id" name="alternative-cache-name">
    <wlevs:caching-system ref="caching-system-id"/>
</wlevs:cache>
...
<wlevs:adapter id="myAdapter" provider="myProvider">
    <wlevs:instance-property name="map" ref="cache-id"/>
</wlevs:adapter>
```

In the example, the `ref` attribute of `wlevs:instance-property` references the `id` value of the `wlevs:cache` element. Oracle CEP automatically injects the cache, implemented as a `java.util.Map`, into the adapter.

In the adapter Java source, add a `setMap` (`Map`) method with the code that implements whatever you want the adapter to do with the cache:

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

```
import java.util.Map;
public class MyAdapter implements Runnable, Adapter, EventSource, SuspendableBean {
    ...
    public void setMap (Map map) {...}
}
```

12.8 Accessing a Cache From a Business POJO

A business POJO, configured as a standard Spring bean in the EPN assembly file, can be injected with a cache using the standard Spring mechanism for referencing another bean. In this way the POJO can view and manipulate the cache. A cache bean implements the `java.util.Map` interface which is what the business POJO uses to access the injected cache. A cache bean can also implement a vendor-specific sub-interface of `java.util.Map`, but for portability it is recommended that you implement `Map`.

First, the configuration of the business POJO in the EPN assembly file must be updated with a `property` child element, as shown in the following example based on the `Output` bean of the FX example (see "HelloWorld Example" in the *Oracle Complex Event Processing Getting Started*):

```
<wlevs:caching-system id="caching-system-id"/>
...
<wlevs:cache id="cache-id" name="alternative-cache-name">
    <wlevs:caching-system ref="caching-system-id"/>
</wlevs:cache>
...
<bean class="com.bea.wlevs.example.helloworld.HelloWorldBean">
    <property name="map" ref="cache-id"/>
</bean>
```

In the example, the `ref` attribute of the `property` element references the `id` value of the `wlevs:cache` element. Oracle CEP automatically injects the cache, implemented as a `java.util.Map`, into the business POJO bean.

In the business POJO bean Java source, add a `setMap (Map)` method with the code that implements whatever you want the POJO to do with the cache:

```
package com.bea.wlevs.example.helloworld;
...
import java.util.Map;
public class HelloWorldBean implements EventSink {
    ...
    public void setMap (Map map) {...}
}
```

12.9 Accessing a Cache From an Oracle CQL User-Defined Function

In addition to standard event streams, Oracle CQL rules can also invoke the member methods of a user-defined function.

These user-defined functions are implemented as standard Java classes and are declared in the component configuration file of the Oracle CQL processor, as shown in the following example:

```
<bean id="orderFunction" class="orderFunction-impl-class"/>
```

The processor in which the relevant Oracle CQL rule runs must then be injected with the user-defined function using the `wlevs:function` child element, referencing the Spring 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 CEP automatically injects the cache, implemented as a `java.util.Map`, into the user-defined function.

In the user-defined function's Java source, add a `setMap` (`Map`) method with the code that implements whatever you want the function to do with the cache:

```

package com.bea.wlevs.example;
...
import java.util.Map;
public class OrderFunction {
  ...
  public void setMap (Map map) {...}
}

```

For more information on user-defined functions, see "Functions: User-Defined" in the *Oracle Complex Event Processing CQL Language Reference*.

12.10 Accessing a Cache From an EPL User-Defined Function

In addition to standard event streams, EPL rules can also invoke the member methods of a user-defined function.

These user-defined functions are implemented as standard Java classes and are declared in the EPN assembly file using the standard Spring bean tags, as shown in the following example:

```
<bean id="orderFunction" class="orderFunction-impl-class"/>
```

The processor in which the relevant EPL rule runs must then be injected with the user-defined function using the `wlevs:function` child element, referencing the Spring with the `ref` attribute:

```
<wlevs:processor id="tradeProcessor">
  <wlevs:function ref="orderFunction"/>
</wlevs:processor>
```

The following EPL rule, assumed to be configured for the `tradeProcessor` processor, shows how to invoke the `existsOrder` method of the `orderFunction` user-defined function:

```
INSERT INTO InstitutionalOrder
SELECT er.orderKey AS key, er.symbol AS symbol, er.shares as cumulativeShares
FROM ExecutionRequest er RETAIN 8 HOURS WITH UNIQUE KEY
WHERE NOT orderFunction.existsOrder(er.orderKey)
```

You can also configure the user-defined function to access a cache by injecting the function with a cache using the standard Spring mechanism for referencing another bean. A cache bean implements the `java.util.Map` interface which is what the user-defined function uses to access the injected cache.

First, the configuration of the user-defined function in the EPN assembly file must be updated with a `wlevs:property` child element, as shown in the following example:

```
<wlevs:caching-system id="caching-system-id"/>
...
<wlevs:cache id="cache-id" name="alternative-cache-name">
  <wlevs:caching-system ref="caching-system-id"/>
</wlevs:cache>
...
<bean id="orderFunction" class="orderFunction-impl-class">
  <wlevs:property name="cache" ref="cache-id"/>
</bean>
```

In the example, the `ref` attribute of the `wlevs:property` element references the `id` value of the `wlevs:cache` element. Oracle CEP automatically injects the cache, implemented as a `java.util.Map`, into the user-defined function.

In the user-defined function's Java source, add a `setMap (Map)` method with the code that implements whatever you want the function to do with the cache:

```
package com.bea.wlevs.example;
...
import java.util.Map;
public class OrderFunction {
  ...
  public void setMap (Map map) {...}
}
```

For more information on user-defined functions, see "User-Defined Functions" in the *Oracle Complex Event Processing EPL Language Reference*.

12.11 Accessing a Cache Using JMX

At runtime, you can access a cache programatically using JMX and the MBeans that Oracle CEP deploys for the caching systems and caches you define.

This section describes:

- [Section 12.11.1, "How to Access a Cache With JMX Using Oracle CEP Visualizer"](#)
- [Section 12.11.2, "How to Access a Cache With JMX Using Java"](#)

For more information, "Configuring JMX for Oracle CEP" in the *Oracle Complex Event Processing Administrator's Guide*

12.11.1 How to Access a Cache With JMX Using Oracle CEP Visualizer

The simplest and least error-prone way to access a caching system or cache with JMX is to use the Oracle CEP Visualizer.

For more information, see "Server and Domain Tasks" in the *Oracle Complex Event Processing Visualizer User's Guide*.

12.11.2 How to Access a Cache With JMX Using Java

The simplest and least error-prone way to access a caching system or cache with JMX is to use the Oracle CEP Visualizer (see [Section 12.11.1, "How to Access a Cache With JMX Using Oracle CEP Visualizer"](#)). Alternatively, you can access a caching system or cache with JMX using Java code that you write.

Oracle CEP creates a `StageMBean` for each cache that your application uses as a stage. The `Type` of this MBean is `Stage`.

To access a cache with JMX using Java:

1. Connect to the JMX service that Oracle CEP server provides.

For more information, see "Configuring JMX for Oracle CEP" in the *Oracle Complex Event Processing Administrator's Guide*

2. Get a list of cache `StageMBean` using either of:

- `CachingSystemMBean.getCacheMBeans()`
- `ApplicationMBean.getStageMBeans()`

3. Get the `ObjectName` for a given `StageMBean` that represents a cache in your caching system:

```
ObjectName cacheName = ObjectName.getInstance (
    'com.bea.wlevs:Name =
newCache,Type=Stage,CachingSystem=newCachingSystem,Application=provider'
);
```

4. Get a proxy instance for the `StageMBean` with this `ObjectName`:

```
StageMBean cache = (StageMBean) MBeanServerInvocationHandler.newProxyInstance(
    server, cacheName, StageMBean.class, false
);
```

5. Use the methods of the `StageMBean` to access the cache.

Configuring Event Record and Playback

This section contains information on the following subjects:

- [Section 13.1, "Overview of Configuring Event Record and Playback"](#)
- [Section 13.2, "Configuring Event Record and Playback in Your Application"](#)
- [Section 13.3, "Creating a Custom Event Store Provider"](#)

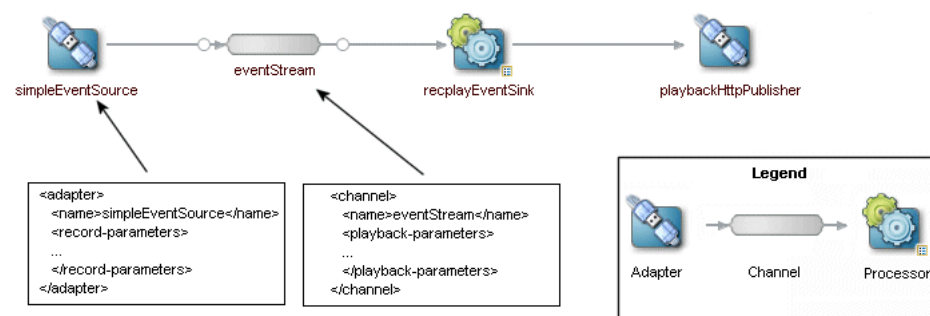
13.1 Overview of Configuring Event Record and Playback

Oracle CEP event repository feature allows you to persist the events that flow out of a component of the event processing network (EPN) to a store, such as a database table, and then play them back at a later stage or explicitly query the events from a component such as an event bean.

A typical use case of this feature is the ability to debug a problem with a currently running application. If you have been recording the events at a node in the EPN when the problem occurred, you can later playback the same list of events to recreate the problem scenario for debugging purposes.

The following graphic shows the EPN of the Event Record and Playback example and demonstrates at what point events are recorded and where they are played back. The `simpleEventSource` adapter has been configured to record events; as indicated, the record happens as events flow *out* of the adapter. The `eventStream` channel has been configured to playback events; as indicated, the playback happens at the point where events flow *into* the channel.

Figure 13–1 Configuring Record and Playback in an EPN



This section describes:

- [Section 13.1.1, "Storing Events in the Persistent Event Store"](#)
- [Section 13.1.2, "Recording Events"](#)

- [Section 13.1.3, "Playing Back Events"](#)
- [Section 13.1.4, "Querying Stored Events"](#)
- [Section 13.1.5, "Record and Playback Example"](#)

13.1.1 Storing Events in the Persistent Event Store

When you record events, Oracle CEP server stores them in a persistent event store. You can use the persistent event store that Oracle CEP server provides or define your own:

- [Section 13.1.1.1, "Default Persistent Event Store"](#)
- [Section 13.1.1.2, "Custom Persistent Event Store"](#)
- [Section 13.1.1.3, "Persistent Event Store Schema"](#)

13.1.1.1 Default Persistent Event Store

By default, Oracle CEP uses a Berkeley DB instance bundled with the Oracle CEP 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 CEP 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 CEP (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 13.2.1, "Configuring an Event Store for Oracle CEP Server"](#) describes.

13.1.1.2 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 13.3, "Creating a Custom Event Store Provider."](#)

13.1.1.3 Persistent Event Store Schema

You do not create the actual database schema used to store the recorded events. Oracle CEP 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 13.2.5, "Description of the Berkeley Database Schema"](#).

13.1.2 Recording Events

You can configure recording for any component in the event processing network (EPN) that produces events: processors, adapters, streams, and event beans. Processors and streams always produce events; adapters and event beans must implement the `EventSource` interface. Additionally, you can configure that events from different components in the EPN be stored in different persistent stores, or that all events go to the same store. Note that only events that are outputted by the component are recorded.

You enable the recording of events for a component by updating its configuration file and adding the `record-parameters` element. Using the child elements of `record-parameters`, you specify the event store to which the events are recorded, an initial time period when recording should take place, the list of event types you want to store, and so on.

After you deploy the application and events start flowing through the network, recording begins either automatically because you configured it to start at a certain time or because you dynamically start it using administration tools. For each component you have configured for recording, Oracle CEP stores the events that flow out of it to the appropriate store along with a timestamp of the time it was recorded.

13.1.3 Playing Back Events

You can configure playback for any component in the event processing network (EPN): processors, adapters, streams, and event beans. Typically the playback component is a node later in the network than the node that recorded the events.

You enable the playback of events for a component by updating its configuration file and adding the `playback-parameters` element. Using the child elements of `playback-parameters`, you specify the event store from which the events are played back, the list event types you want to play back (by default all are played back), the time range of the recorded events you want to play back, and so on. By default, Oracle CEP plays back the events in a time accurate manner; however, you can also configure that the events get played back either faster or slower than they originally flowed out of the component from which they were recorded.

After you deploy the application and events start flowing through the network, you must start the playback by using the administration tools (Oracle CEP Visualizer or `wlevs.Admin`). Oracle CEP reads the events from the appropriate persistent store and inserts them into the appropriate place in the EPN.

It is important to note that when a component gets a playback event, it looks exactly like the original event. Additionally, a component later in the network has been configured to record events, then Oracle CEP records the playback events as well as the "real" events.

For more information, see:

- "Recording and Playing Back Events Flowing Through an EPN" in the *Oracle Complex Event Processing Visualizer User's Guide*
- "Commands for Controlling Event Record and Playback" in the *Oracle Complex Event Processing Administrator's Guide*

13.1.4 Querying Stored Events

You can use the event store API to query a store for past events given a record time range and the component from which the events were recorded. The actual query you use depends on the event repository provider; for example, you would use Oracle

CQL or EPL for the default persistent event store provider included with Oracle CEP. You can also use these APIs to delete old events from the event store.

13.1.5 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_11.1.1\samples\source\applications\recplay` directory, where `ORACLE_CEP_HOME` refers to the main Oracle CEP installation directory, such as `d:\oracle_cep`.

For details about running and building the example, see "Event Record and Playback Example" in the *Oracle Complex Event Processing Getting Started*.

13.2 Configuring Event Record and Playback in Your Application

Depending on how you are going to use the event repository, there are different tasks that you must perform, as described in the following procedure that in turn point to sections with additional details.

Note: It is assumed in this section that you have already created an Oracle CEP application along with its component configuration file(s) and that you want to update the application so that components record or playback events. If you have not, refer to [Chapter 1, "Overview of Creating Oracle CEP Applications,"](#) for details.

To configure record and playback of events in your application:

1. Optionally configure the Berkeley database event store for your Oracle CEP 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 13.2.1, "Configuring an Event Store for Oracle CEP Server."](#)

2. Configure a component in your EPN to record events by updating the component's configuration file.

The component can be a processor, adapter, channel, or event bean. Only events flowing out of the component are recorded.

See [Section 13.2.2, "Configuring a Component to Record Events."](#)

3. Configure a component in your EPN to playback events by updating the component's configuration file.

The component can be a processor, adapter, channel, or event bean. Only components that are also event sinks can playback events; events are played to the input side of the component.

See [Section 13.2.3, "Configuring a Component to Playback Events."](#)

4. Redeploy your application for the changes to take effect.
5. If you have not specified an explicit start and end time for recording events, you must use Oracle CEP Visualizer or `wlevs.Admin` to start recording. You must always use these administration tools to start and end the playback of events.

See [Section 13.2.4, "Starting and Stopping the Record and Playback of Events."](#)

13.2.1 Configuring an Event Store for Oracle CEP Server

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 13.3, "Creating a Custom Event Store Provider"](#).

To configure an event store for Oracle CEP server:

1. Stop your Oracle CEP server instance, if it is running.
2. Using your favorite XML editor, open the server's `config.xml` file for edit.

The `config.xml` file is located in the `DOMAIN_DIR/servername/config` directory of your server, where `DOMAIN_DIR` refers to the domain directory, such as `/oracle_cep/user_projects/domains/myDomain` and `servername` refers to the name of your server, such as `defaultserver`.

3. Edit the `bdb-config` element to the `config.xml` file.

[Example 13–1](#) shows a fully configured `bdb-config` element.

Example 13–1 `bdb-config` Element

```
<bdb-config>
  <db-env-path>bdb</db-env-path>
  <cache-size>1000</cache-size>
</bdb-config>
```

[Table 13–1](#) lists the child elements of `bdb-config` that you can specify.

Table 13–1 Child Elements of `bdb-config`

Child Element	Description
<code>db-env-path</code>	Specifies the subdirectory in which Oracle CEP server creates Berkeley database instances relative to the 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>
<code>cache-size</code>	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"> ▪ http://www.oracle.com/technology/documentation/berkeley-db/je/GettingStartedGuide/cachesize.html. ▪ http://www.oracle.com/technology/documentation/berkeley-db/je/java/com/sleepycat/je/EnvironmentMutableConfig.html#setCacheSize(long) Default: <code>je.maxMemoryPercent * JVM maximum memory</code>

4. Restart your Oracle CEP server instance.

13.2.2 Configuring a Component to Record Events

You can configure any processor, adapter, channel, or event bean in your application to record events. As with all other component configuration, you specify that a component records events by updating its configuration file. For general information about these configuration files, see [Section 1.1.5, "Component Configuration Files."](#)

This section describes the main steps to configure a component to record events. For simplicity, it is assumed in the procedure that you are configuring an adapter to record events and that you have already created its component configuration file.

See [Section B.2, "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"?>
  <n1:config xmlns:n1="http://www.bea.com/ns/wlevs/config/application"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
    <adapter>
      <name>simpleEventSource</name>
      <record-parameters>
        ...
      </record-parameters>
      ...
    </adapter>
    ...
  </n1:config>
```

Add child elements to `record-parameters` to specify the name of the event store provider, the events that are stored, the start and stop time for recording, and so on. For example:

```
<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 13–2](#) lists the child elements of `record-parameters` that you can specify. Only `dataset-name` is required.

Table 13–2 Child Elements of record-parameters

Child Element	Description
dataset-name	Specifies the group of data that the user wants to group together. In the case of BDB provider, the dataset name will be used as the database environment in Berkeley database. In the case of the Oracle RDBMS-based provider, it specifies the database area, or schema, in which the tables that store the recorded events are created. When configuring the Oracle RDBMS-based provider, you are required to specify this element.

Table 13–2 (Cont.) Child Elements of record-parameters

Child Element	Description
event-type-list	<p>Specifies the event types that are recorded to the event store. If this element is not specified, then Oracle CEP 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><event-type-list> <event-type>EventOne</event-type> <event-type>EventTwo</event-type> </event-type-list></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 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><time-range> <start>2010-01-20T05:00:00</start> <end>2010-01-20T18:00:00</end> </time-range></pre> <p>For complete details of the XML Schema <code>dateTime</code> format, see http://www.w3.org/TR/xmlschema-2/#dateTime-lexical-representation.</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 CEP 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><time-range-offset> <start>2010-01-20T05:00:00</start> <duration>03:00:00</duration> </time-range-offset></pre> <p>For complete details of the XML Schema <code>dateTime</code> format, see http://www.w3.org/TR/xmlschema-2/#dateTime-lexical-representation.</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 CEP 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>

Table 13–2 (Cont.) Child Elements of record-parameters

Child Element	Description
batch-size	Specifies the number of events that Oracle CEP picks up in a single batch from the event buffer to write the event store. Default value is 1000.
batch-time-out	Specifies the number of seconds that Oracle CEP waits for the event buffer window to fill up with the batch-size number of events before writing to the event store. Default value is 60
max-size	If specified, Oracle CEP uses a stream when writing to the event store, and this element specifies the size of the stream, with non-zero values indicating asynchronous writes. Default value is 1024.
max-threads	If specified, Oracle CEP uses a stream when writing to the event store, and this element specifies the maximum number of threads that will be used to process events for this stream. Setting this value has no effect when max-size is 0. The default value is 1.

13.2.3 Configuring a Component to Playback Events

You can configure any processor, adapter, channel, or event bean in your application to playback events, although the component must be a node downstream of the recording component so that the playback component will actually receive the events and play them back. As with all other component configuration, you specify that a component plays back events by updating its configuration file. For general information about these configuration files, see [Section 1.1.5, "Component Configuration Files."](#)

This section describes the main steps to configure a component to play back events. For simplicity, it is assumed in the procedure that you are configuring a channel to playback events from a node upstream in the EPN that has recorded events, and that you have already created the channel's configuration file.

See for the complete XSD Schema that describes the event playback configuration file elements.

Using your favorite XML editor, open the component configuration XML file and add a playback-parameters child element to the component you want to configure to playback events. For example, to configure a channel called eventStream:

```
<?xml version="1.0" encoding="UTF-8"?>
  <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 13–3 lists the child elements of `playback-parameters` that you can specify. Only `dataset-name` is required.

Table 13–3 Child Elements of `playback-parameters`

Child Element	Description
<code>dataset-name</code>	<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>
<code>event-type-list</code>	<p>Specifies the event types that are played back from the event store. If this element is not specified, then Oracle CEP plays back <i>all</i> event types.</p> <p>Use the <code>event-type</code> child component to list one or more events, such as:</p> <pre> <event-type-list> <event-type>EventOne</event-type> <event-type>EventTwo</event-type> </event-type-list> </pre> <p>When configuring the Oracle RDBMS-based provider, you are required to specify this element.</p>
<code>time-range</code>	<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 <code>start</code> child element to specify a start time and an <code>end</code> child element to specify the end time. 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 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> <time-range> <start>2010-01-20T05:00:00</start> <end>2010-01-20T18:00:00</end> </time-range> </pre> <p>For complete details of the XML Schema <code>dateTime</code> format, see http://www.w3.org/TR/xmlschema-2/#dateTime-lexical-representation.</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 CEP 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>

Table 13–3 (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><time-range-offset> <start>2010-01-20T05:00:00</start> <duration>03:00:00</duration> </time-range-offset></pre> <p>For complete details of the XML Schema <code>dateTime</code> format, see http://www.w3.org/TR/xmlschema-2/#dateTime-lexical-representation.</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 CEP 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 CEP uses a stream when playing back events from the event store, and this element specifies the size of the stream, with non-zero values indicating asynchronous writes.</p> <p>Default value is 1024.</p>
max-threads	<p>If specified, Oracle CEP uses a stream when playing back events from the event store, and this element specifies the maximum number of threads that will be used to process events for this stream. Setting this value has no effect when <code>max-size</code> is 0.</p> <p>The default value is 1.</p>

13.2.4 Starting and Stopping the Record and Playback of Events

After you configure the record and playback functionality for the components of an application, and you deploy the application to Oracle CEP, the server starts to record events only if you specified an explicit start/stop time in the initial configuration.

For example, if you included the following element in a component configuration:

```
<time-range>
  <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 CEP Visualizer or `wlevs.Admin`. You also use these tools to dynamically start and stop the recording of events.

For more information, see:

- "Recording and Playing Back Events Flowing Through an EPN" in the *Oracle Complex Event Processing Visualizer User's Guide*
- "Commands for Controlling Event Record and Playback" in the *Oracle Complex Event Processing Administrator's Guide*

Visualizer and `wlevs.Admin` use managed beans (MBeans) to dynamically start and stop event recording and playback, as well as manage the event store configuration. A managed bean is a Java bean that provides a Java Management Extensions (JMX) interface. JMX is the Java EE solution for monitoring and managing resources on a network. You can create your own administration tool and use JMX to manage event store functionality by using the `com.bea.wlevs.management.configuration.StageMBean`.

For more information, see:

- "Configuring JMX for Oracle CEP" in the *Oracle Complex Event Processing Administrator's Guide*
- *Oracle Fusion Middleware Java API Reference for Oracle Complex Event Processing*

13.2.5 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 CEP 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 13–2](#) shows.

Example 13–2 Default `bdb-config` Element

```
<bdb-config>
  <db-env-path>bdb</db-env-path>
</bdb-config>
```

If your `dataset-name` is `test1`, then Oracle CEP 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 CEP (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 CEP 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.

13.3 Creating a Custom Event Store Provider

Oracle CEP provides an event store API that you can use to create a custom event store provider. Oracle provides an RDBMS-based implementation for storing events in a relational database, or one that supports JDBC connections. If you want to store events in a different kind of database, or for some reason the Oracle RDBMS provider is not adequate for your needs, then you can create your own event store provider using the event store API.

The event store API is in the `com.bea.wlevs.eventstore` package; the following list describes the most important interfaces:

- `EventStore`—Object that represents a single event store. The methods of this interface allow you to persist events to the store and to query the contents of the store using a provider-specific query.
- `EventStoreManager`—Manages event stores. Only one instance of the `EventStoreManager` ever exists on a given Oracle CEP server, and this instance registers itself in the OSGi registry so that event store providers can in turn register themselves with the event store manager. You use this interface to find existing event stores, create new ones, get the provider for a given event store, and register an event provider. The event store manager delegates the actual work to the event store provider.
- `EventStoreProvider`—Underlying repository that provides event store services to clients.

For more information, see the *Oracle Fusion Middleware Java API Reference for Oracle Complex Event Processing*.

Part IV

Extending the Oracle CEP Event Processing Network

Part IV contains the following chapters:

- [Chapter 14, "Configuring Custom Adapters,"](#)
- [Chapter 15, "Configuring Custom Event Beans,"](#)
- [Chapter 16, "Configuring Custom Spring Beans,"](#)
- [Chapter 17, "Configuring Web Services"](#)
- [Chapter 18, "Configuring Applications With Data Cartridges"](#)
- [Chapter 19, "Extending Component Configuration,"](#)

Configuring Custom Adapters

This chapter describes how to code and register custom adapters, including:

- [Section 14.1, "Overview of Custom Adapters"](#)
- [Section 14.2, "Implementing a Custom Adapter"](#)
- [Section 14.3, "Passing Login Credentials from an Adapter to a Data Feed Provider"](#)
- [Section 14.4, "Configuring the Custom Adapter EPN Assembly File"](#)
- [Section 14.5, "Configuring the Custom Adapter Component Configuration File"](#)

14.1 Overview of Custom Adapters

One of the main roles of an adapter is to convert data coming from some channel, such as a market data feed, into Oracle CEP events. These events are then passed to other components in the application, such as processors. An adapter is usually the entry point to an Oracle CEP application. An adapter can also be the exit point of an application so that it receives events from an intermediate component, converts the data into something that an external application can read, and then sends it out.

"Foreign Exchange (FX) Example" in the *Oracle Complex Event Processing Getting Started* shows three adapters that read in data from currency data feeds and then pass the data, in the form of a specific event type, to the processors, which are the next components in the network.

You can create adapters of different types, depending on the format of incoming data and the technology you use in the adapter code to do the conversion. The most typical types of adapters are those that:

- Use a data vendor API, such as Reuters, Wombat, or Bloomberg.
- Convert incoming JMS messages using standard JMS APIs.
- Use other messaging systems, such as TIBCO Rendezvous.
- Use a socket connection to the customer's own data protocol.

Adapters are Java classes that implement specific Oracle CEP interfaces. You register the adapter classes in the EPN assembly file that describes your entire application.

You can optionally change the default configuration of the adapter, or even extend the configuration and add new configuration elements and attributes.

There are two ways to pass configuration data to the adapter; the method you chose depends on whether you want to dynamically change the configuration after deployment:

- If you are *not* going to change the configuration data after the adapter is deployed, then you can configure the adapter in the EPN assembly file.
- If, however, you do want to be able to dynamically change the configuration elements, then you should put this configuration in the adapter-specific component configuration file.

This section describes:

- [Section 14.1.1.1, "Custom Adapter Event Sources and Event Sinks"](#)
- [Section 14.1.2, "Custom Adapter Factories"](#)
- [Section 14.1.3, "Single and Multi-threaded Adapters"](#)

14.1.1 Custom Adapter Event Sources and Event Sinks

Adapters can be event sources, event sinks, or both. Event sources generate events, event sinks receive events.

14.1.1.1 Custom Adapters as Event Sources

You specify that an adapter component in your EPN is an event source by implementing the `com.bea.wlevs.ede.api.StreamSource` or `RelationSource` API.

The implementation class of a custom adapter that is an event source may be specified as private to the application or as an advertised OSGI service re-usable by other applications.

You register adapters in the EPN assembly file using the `wlevs:adapter` element. For example:

```
<wlevs:adapter id="myAdapterSource" class="com.acme.MySourceAdapter">
</wlevs:adapter>
```

In this example, the Java class `MySourceAdapter.java` implements the `com.bea.wlevs.ede.api.StreamSource` or `RelationSource` API.

For more information, see [Section 14.2.2.1, "Implementing a Custom Adapter as an Event Source"](#).

14.1.1.2 Custom Adapters as Event Sinks

The functionality of custom adapters as event sinks is very similar to that of event sources except that custom adapter event sinks must implement the `com.bea.wlevs.ede.api.StreamSink` or `RelationSink` API.

Event sinks are not active which means that if they implement the `Runnable` interface, Oracle CEP does not run them in a separate thread.

You reference event sinks in the EPN assembly file using the `wlevs:listener` element:

```
<wlevs:adapter id="myAdapterSink" class="com.acme.MySinkAdapter">
</wlevs:adapter>

<wlevs:channel id="myStream" >
  <wlevs:listener ref="myAdapterSink" />
</wlevs:channel>
```

In this example, the Java class `MySinkAdapter.java` implements the `com.bea.wlevs.ede.api.StreamSink` or `RelationSink` API.

For more information, see [Section 14.2.2.2, "Implementing a Custom Adapter as an Event Sink"](#).

14.1.2 Custom Adapter Factories

If your adapter is going to be used only by a single Oracle CEP application, then you do not need to create a factory. However, if multiple applications are going to use the same adapter, then you should also program a factory. In this case, every application gets its own instance of the adapter.

Adapter factories must implement the `com.bea.wlevs.ede.api.Factory` interface. This interface has a single method, `create`, that you implement to create an adapter or event bean instance.

You register factories in the EPN assembly file using the `wlevs:factory` element:

```
<wlevs:factory provider-name="myprovider" class="my.Implementation"/>
```

Note that if you need to specify service properties, then you must use the `osgi:service` element to register the factory.

14.1.3 Single and Multi-threaded Adapters

By default, an adapter is single-threaded. That is, an adapter uses a single thread to read messages from its data source.

When an adapter is single-threaded, event order is guaranteed.

To improve scalability, you can configure an adapter to use multiple threads to read from its data source. 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 an adapter is multi-threaded, event order is not guaranteed.

For more information, see:

- [Section F.44, "work-manager"](#)
- [Section 22.2.1, "EventPartitioner"](#)

14.2 Implementing a Custom Adapter

You implement a custom adapter in a Java class that you create. To create this Java class, you can either:

- Create a fully functioning empty adapter using Ant and then modify this generated class to suit your application as [Section 14.2.1, "How to Implement a Custom Adapter Using Ant"](#) describes.
- Create the adapter Java class manually as [Section 14.2.2, "How to Implement a Custom Adapter Manually"](#) describes.

For more information, see [Section 14.1, "Overview of Custom Adapters"](#).

14.2.1 How to Implement a Custom Adapter Using Ant

The following procedure describes how to use the Oracle CEP Ant task `TBD` to generate a fully functioning empty adapter.

To implement a custom adapter using Ant:

- 1.
- 2.
3. Optionally program a factory class.
You only need to do this if many applications are going to use the custom adapter.
See [Section 14.2.3, "Implementing a Custom Adapter Factory."](#)
4. Optionally extend the component configuration of the custom adapter if the default component configuration is not adequate.
See [Chapter 19, "Extending Component Configuration."](#)
5. Configure the custom adapter.
For more information, see:
 - [Section 14.4, "Configuring the Custom Adapter EPN Assembly File."](#)
 - [Section 14.5, "Configuring the Custom Adapter Component Configuration File."](#)

14.2.2 How to Implement a Custom Adapter Manually

The following procedure describes the typical steps for manually creating a custom adapter.

Note: The following procedure assumes that the custom adapter is bundled in the same application JAR file that contains the other components of the event network, such as the processor, streams, and business logic POJO. If you want to bundle the custom adapter in its own JAR file so that it can be shared among multiple applications, see [Section 24.2.4.1, "How to Assemble a Custom Adapter in its Own Bundle."](#)

To implement a custom adapter:

1. Program the custom adapter Java class.
If your custom adapter is an event source, see [Section 14.2.2.1, "Implementing a Custom Adapter as an Event Source."](#)
If your custom adapter is an event sink, see [Section 14.2.2.2, "Implementing a Custom Adapter as an Event Sink."](#)
If your custom adapter is both an event source and event sink, then see both sections.
If your custom adapter must authenticate itself with a data feed provider, see [Section 14.3, "Passing Login Credentials from an Adapter to a Data Feed Provider."](#)
2. Optionally program a factory class.
You only need to do this if many applications are going to use the custom adapter.
See [Section 14.2.3, "Implementing a Custom Adapter Factory."](#)
3. Optionally extend the component configuration of the custom adapter if the default component configuration is not adequate.
See [Chapter 19, "Extending Component Configuration."](#)

4. Configure the custom adapter.

For more information, see:

- [Section 14.4, "Configuring the Custom Adapter EPN Assembly File."](#)
- [Section 14.5, "Configuring the Custom Adapter Component Configuration File."](#)

14.2.2.1 Implementing a Custom Adapter as an Event Source

This section describes how to create a custom adapter that acts as an event source because it receives incoming data and generates events that it sends to the next component in the EPN.

The inbound custom adapter class typically reads the stream of incoming data, such as from a market data feed, converts it into an Oracle CEP event type that is understood by the rest of the application, and sends the event to the next component in the network.

The following example shows the custom adapter class of the Spatial sample; see the explanation after the example for coding guidelines that correspond to the Java code in bold.

```
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;

import oracle.spatial.geometry.JGeometry;
import com.oracle.cep.cartridge.spatial.Geometry;

public class BusStopAdapter implements RunnableBean, StreamSource, StreamSink
{
    private final static String APP_NAME          = "spatial_sample";
    ...
    private String          m_eventTypeName;
    private EventType      m_eventType;
    private StreamSender    m_eventSender;
    ...
    public BusStopAdapter()
    {
        super();
    }
    ...
    @Service(filter = EventTypeRepository.SERVICE_FILTER)
    public void setEventTypeRepository(EventTypeRepository etr)
    {
        m_etr = etr;
    }
}
```

```
public void setShow(boolean b)
{
    m_show = b;
}

public void run()
{
    m_stopped = false;

    // Need to store running thread so that it can be interrupted in case of
    // suspend.
    m_runningThread = Thread.currentThread();

    if (m_ettr == null)
    {
        throw new RuntimeException("EventTypeRepoitory is not set");
    }

    m_eventType = m_ettr.getEventType(m_eventTypeName);
    if (m_eventType == null)
    {
        throw new RuntimeException("EventType(" + m_eventType + ") is not found.");
    }

    s_logger.info("fileSource " + m_filePath);

    if (m_initialDelay > 0)
    {
        try
        {
            s_logger.info("Sleeping for : " + m_initialDelay);
            Thread.sleep(m_initialDelay);
        } catch (InterruptedException e)
        {
            e.printStackTrace();
        }
    }

    synchronized (this)
    {
        while (!m_start)
        {
            try
            {
                this.wait();
            } catch (InterruptedException e)
            {
            }
        }
    }
    BufferedReader reader = null;

    System.out.println("Sending " + m_eventType + " from " + m_filePath);
    while ((m_repeat != 0) && (!m_stopped))
    {
        try
        {
            {
                s_logger.info("Starting fileSource " + m_filePath);
                m_lineno = 0;
                reader = new BufferedReader(new FileReader(m_filePath));
            } catch (Exception e)
            {
                s_logger.warn(e.toString());
                m_stopped = true;
                break;
            }
        }
    }
}
```

```

    }
    while (!isStopped())
    {
        try
        {
            Object ev = null;
            ev = readLine(reader);
            if (ev == null)
            {
                reader.close();
                break;
            }
            m_eventSender.sendInsertEvent(ev);
            if (m_show)
            {
                System.out.println(ev.toString());
            }
        } catch (Exception e)
        {
            s_logger.fatal("Failed to get tuple from " + m_filePath + ":"
                + m_lineno + "\n" + e.toString() + "\n");
            m_stopped = true;
            break;
        }
        if (m_sleep > 0)
        {
            try
            {
                synchronized (this)
                {
                    // We need to save thread so that it can be interrupted in case of
                    // suspend.
                    wait(m_sleep);
                }
            } catch (InterruptedException e)
            {
                s_logger.warn(e);
                break;
            }
        }
    } // while
    if (m_repeat > 0)
        m_repeat--;
} // while
s_logger.info("FileAdaptor " + hashCode() + " stopped. " + m_filePath);
}

public void setEventSender(StreamSender sender)
{
    m_eventSender = sender;
}

...
}

```

Follow these guidelines when programming the adapter Java class; code snippets of the guidelines are shown in bold in the preceding example:

- **Import the interfaces and classes of the Oracle CEP API:**

```

import com.bea.wlevs.ede.api.StreamSender;
import com.bea.wlevs.ede.api.StreamSource;
import com.bea.wlevs.ede.api.RunnableBean;

```

Because the adapter is an event source it must implement the `StreamSource` interface. If you want the adapter to run in a thread, also implement

`RunnableBean`. The `StreamSender` interface sends event types to the next component in your application network. For full details of these APIs, see *Oracle Fusion Middleware Java API Reference for Oracle Complex Event Processing*.

- The adapter class must implement the `StreamSource` and `RunnableBean` interfaces because it is an event source and will run in its own thread:

```
public class BusStopAdapter implements RunnableBean, StreamSource, StreamSink
```

The `StreamSource` interface provides the `StreamSender` that you use to send events.

- Because the adapter implements the `RunnableBean` interface, your adapter must then implement the `run` method:

```
public void run() {...
```

This is where you should put the code that reads the incoming data, such as from a market feed, and convert it into an Oracle CEP event type, and then send the event to the next component in the network. Refer to the documentation of your data feed provider for details on how to read the incoming data. See [Section 24.2.2.2, "Accessing Third-Party JAR Files"](#) for information about ensuring you can access the vendor APIs if they are packaged in a third-party JAR file.

In the Spatial example, the adapter itself generates the incoming data using the `readLine` protected method. This is just for illustrative purposes and is not a real-world scenario.

For more information, see [Chapter 2.1.4, "Creating Oracle CEP Event Types"](#).

- Because your adapter implements `StreamSource`, you must implement the `setEventSender` method, which passes in the `StreamSender` that you use to send events:

```
public void setEventSender(StreamSender sender) { ...
```

- If, as is typically the case, your adapter implements `SuspendableBean`, you must implement the `suspend` method that stops the adapter when, for example, the application is undeployed:

```
public synchronized void suspend() throws Exception { ...
```

14.2.2.2 Implementing a Custom Adapter as an Event Sink

The following sample code shows a Spring bean from HelloWorld application that acts as an event sink; see the explanation after the example for the code shown in bold:

The following example shows the custom adapter class of the Spatial sample; see the explanation after the example for coding guidelines that correspond to the Java code in bold.

```
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;

import oracle.spatial.geometry.JGeometry;
import com.oracle.cep.cartridge.spatial.Geometry;

public class BusStopAdapter implements RunnableBean, StreamSource, StreamSink
{
    private final static String APP_NAME          = "spatial_sample";
    ...
    private String          m_eventTypeName;
    private EventType      m_eventType;
    private StreamSender    m_eventSender;
    ...
    public BusStopAdapter()
    {
        super();
    }
    ...
    @Service(filter = EventTypeRepository.SERVICE_FILTER)
    public void setEventTypeRepository(EventTypeRepository etr)
    {
        m_etr = etr;
    }

    public void setShow(boolean b)
    {
        m_show = b;
    }

    public void run()
    {
        m_stopped = false;

        // Need to store running thread so that it can be interrupted in case of
        // suspend.
        m_runningThread = Thread.currentThread();

        if (m_etr == null)
        {
            throw new RuntimeException("EventTypeRepository is not set");
        }

        m_eventType = m_etr.getEventType(m_eventTypeName);
        if (m_eventType == null)
        {
            throw new RuntimeException("EventType(" + m_eventType + ") is not found.");
        }

        s_logger.info("fileSource " + m_filePath);

        if (m_initialDelay > 0)
        {
            try
            {
                s_logger.info("Sleeping for : " + m_initialDelay);
                Thread.sleep(m_initialDelay);
            } catch (InterruptedException e)
            {
                e.printStackTrace();
            }
        }
    }
}

```

```
    }

    synchronized (this)
    {
        while (!m_start)
        {
            try
            {
                this.wait();
            } catch (InterruptedException e)
            {
            }
        }
    }
}
BufferedReader reader = null;

System.out.println("Sending " + m_eventType + " from " + m_filePath);
while ((m_repeat != 0) && (!m_stopped))
{
    try
    {
        {
            s_logger.info("Starting fileSource " + m_filePath);
            m_lineno = 0;
            reader = new BufferedReader(new FileReader(m_filePath));
        } catch (Exception e)
        {
            s_logger.warn(e.toString());
            m_stopped = true;
            break;
        }
    }
    while (!isStopped())
    {
        try
        {
            {
                Object ev = null;
                ev = readLine(reader);
                if (ev == null)
                {
                    reader.close();
                    break;
                }
            }
            m_eventSender.sendInsertEvent(ev);
            if (m_show)
            {
                System.out.println(ev.toString());
            }
        } catch (Exception e)
        {
            {
                s_logger.fatal("Failed to get tuple from " + m_filePath + ":"
                    + m_lineno + "\n" + e.toString() + "\n");
                m_stopped = true;
                break;
            }
        }
        if (m_sleep > 0)
        {
            try
            {
                {
                    synchronized (this)
                    {
                        // We need to save thread so that it can be interrupted in case of
                        // suspend.
                        wait(m_sleep);
                    }
                } catch (InterruptedException e)
                {
                    s_logger.warn(e);
                }
            }
        }
    }
}
```

```

        break;
    }
} // while
if (m_repeat > 0)
    m_repeat--;
} // while
s_logger.info("FileAdaptor " + hashCode() + " stopped. " + m_filePath);
}

@Override
public void onInsertEvent(Object event) throws EventRejectedException {
    if (!m_start)
    {
        m_start = true;
        synchronized (this)
        {
            notify();
        }
    }
}
...
}

```

The programming guidelines shown in the preceding example are as follows:

- Your custom adapter must implement the `com.bea.wlevs.ede.api.StreamSink` interface:

```
public class BusStopAdapter implements RunnableBean, StreamSource, StreamSink
```

- The `StreamSink` interface has a single method that you must implement, `onInsertEvent(java.lang.Object)`, which is a callback method for receiving events. The parameter of the method is an `Object` that represents the actual event that the bean received from the component that sent it the event:

```
public void onInsertEvent(Object event)
```

For complete API reference information about the Oracle CEP APIs described in this section, see the *Oracle Fusion Middleware Java API Reference for Oracle Complex Event Processing*.

14.2.3 Implementing a Custom Adapter Factory

Your adapter factory class must implement the `com.bea.wlevs.ede.api.AdapterFactory` interface, which has a single method, `create`, in which you code the creation of your specific adapter class. Event beans implement `Factory`.

The following is a possible adapter factory class for the HelloWorld example:

```

package com.bea.adapter.wlevs.example.helloworld;
import com.bea.wlevs.ede.api.Adapter;
import com.bea.wlevs.ede.api.AdapterFactory;
public class HelloWorldAdapterFactory implements Factory {
    public HelloWorldAdapterFactory() {
    }
    public synchronized Adapter create() throws IllegalArgumentException {
        return new HelloWorldAdapter();
    }
}

```

For full details of these APIs, see the *Oracle Fusion Middleware Java API Reference for Oracle Complex Event Processing*.

14.3 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 (username and password) to the data feed for user authentication.

The simplest, and least secure, way to do this is to hard-code the non-encrypted login credentials in your adapter Java code. However, this method does not allow you to encrypt the password or later change the login credentials without recompiling the adapter Java code.

The following procedures describe a different method that takes these two issues into account. In the procedure, it is assumed that the username to access the data feed is `juliet` and the password is `superSecret`.

You must decide whether you want the login credentials to be configured statically in the EPN assembly file, or dynamically by extending the configuration of the adapter. Configuring the credentials statically in the EPN assembly file is easier, but if the credentials later change you must restart the application for the update to the EPN assembly file to take place. Extending the adapter configuration allows you to change the credentials dynamically without restarting the application, but extending the configuration involves additional steps, such as creating an XSD file and compiling it into a JAXB object.

This section describes:

- [Section 14.3.1, "How to Pass Static Login Credentials to the Data Feed Provider"](#)
- [Section 14.3.2, "How to Pass Dynamic Login Credentials to the Data Feed Provider"](#)
- [Section 14.3.3, "How to Access Login Credentials From an Adapter at Runtime"](#)

For more information, see [Chapter 19, "Extending Component Configuration"](#).

14.3.1 How to Pass Static Login Credentials to the Data Feed Provider

This procedure describes how to pass login credentials that you configure statically in the EPN assembly file.

To pass static credentials to the data feed provider:

1. Open a command window and set your environment as described in "Setting Your Development Environment" in the *Oracle Complex Event Processing Getting Started*.
2. Change to the directory that contains the EPN assembly file for your application.
3. Using your favorite XML editor, edit the EPN assembly file by updating the `wlevs:adapter` element that declares your adapter.

In particular, add two instance properties that correspond to the username and password of the login credentials. For now, specify the cleartext password value; you will encrypt it in a later step. Also add a temporary `password` element whose value is the cleartext password. For example:

```
<wlevs:adapter id="myAdapter" provider="myProvider">
  <wlevs:instance-property name="user" value="juliet"/>
  <wlevs:instance-property name="password" value="superSecret"/>
  <password>superSecret</password>
</wlevs:adapter>
```

4. Save the EPN assembly file.

5. Use the `encryptMSAConfig` command to encrypt the value of the `password` element in the EPN assembly file:

```
prompt> ORACLE_CEP_HOME/ocep_11.1/bin/encryptMSAConfig . epn_assembly_file
msainternal.dat_file
```

where `ORACLE_CEP_HOME` refers to the main directory into which you installed Oracle CEP, such as `d:\oracle_cep`. The second argument refers to the directory that contains the EPN assembly file; because this procedure directs you to change to the directory, the example shows `". "`. The `epn_assembly_file` parameter refers to the name of your EPN assembly file. Finally, the `msainternal.dat_file` parameter refers to the location of the `.msainternal.dat` file associated with your domain; by default this file is located in the `DOMAIN_DIR/servername` directory, where `DOMAIN_DIR` refers to the domain directory such as `/oracle_cep/user_projects/domains/mydomain` and `servername` refers to the server instance.

For more information, see "The `encryptMSAConfig` Command-Line Utility" in the *Oracle Complex Event Processing Administrator's Guide*.

After you run the command, the value of the `password` element of the EPN assembly file will be encrypted.

6. Edit the EPN assembly file:

Copy the encrypted value of the `password` element to the `value` attribute of the `password` instance property.

Remove the `password` element from the XML file.

For example:

```
<wlevs:adapter id="myAdapter" provider="myProvider">
  <wlevs:instance-property name="user" value="juliet"/>
  <wlevs:instance-property name="password"
    value="{Salted-3DES}B7L6nehu7dgPtJTTnTJWRA==" />
</wlevs:adapter>
```

7. Update your adapter Java code to access the login credentials properties you have just configured and decrypt the password.

See [Section 14.3.1, "How to Pass Static Login Credentials to the Data Feed Provider."](#)

8. Edit the `MANIFEST.MF` file of the application and add the `com.bea.core.encrypted` package to the `Import-Package` header. See [Section 24.2.2.1, "Creating the MANIFEST.MF File."](#)
9. Re-assemble and deploy your application as usual. See [Chapter 24, "Assembling and Deploying Oracle CEP Applications."](#)

14.3.2 How to Pass Dynamic Login Credentials to the Data Feed Provider

This procedure describes how to pass login credentials that you configure dynamically by extending the configuration of the adapter.

To pass dynamic login credentials 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 19, "Extending Component Configuration"](#) for detailed instructions.

2. Open a command window and set your environment as described in "Setting Your Development Environment" in the *Oracle Complex Event Processing Getting Started*.
3. Change to the directory that contains the component configuration XML file for your adapter.
4. Using your favorite XML editor, update this component configuration XML file by adding the required login credentials using the `<user>` and `<password>` elements. For now, specify the cleartext password value; you will encrypt it in a later step. For example:

```

<?xml version="1.0" encoding="UTF-8"?>
<myExample:config
  xmlns:myExample="http://www.bea.com/xml/ns/wlevs/example/myExample">
  <adapter>
    <name>myAdapter</name>
    <user>juliet</user>
    <password>superSecret</password>
  </adapter>
</myExample:config>
    
```

5. Save the adapter configuration file.
6. Use the `encryptMSAConfig` command to encrypt the value the `password` element in the adapter configuration file:

```

prompt> ORACLE_CEP_HOME/ocep_11.1/bin/encryptMSAConfig . adapter_config_file
msainternal.dat_file
    
```

where `ORACLE_CEP_HOME` refers to the main directory into which you installed Oracle CEP, such as `d:\oracle_cep`. The second argument refers to the directory that contains the adapter configuration file; because this procedure directs you to change to the directory, the example shows `"."`. The `adapter_config_file` parameter refers to the name of your adapter configuration file. Finally, the `msainternal.dat_file` parameter refers to the location of the `.msainternal.dat` file associated with your domain; by default this file is located in the `DOMAIN_DIR/servername` directory, where `DOMAIN_DIR` refers to the domain directory such as `/oracle_cep/user_projects/domains/mydomain` and `servername` refers to the server instance.

For more information, see "The `encryptMSAConfig` Command-Line Utility" in the *Oracle Complex Event Processing Administrator's Guide*.

After you run the command, the value of the `password` element will be encrypted.

7. Update your adapter Java code to access the login credentials properties you have just configured and decrypt the password.

See [Section 14.3.1, "How to Pass Static Login Credentials to the Data Feed Provider."](#)

8. Edit the `MANIFEST.MF` file of the application and add the `com.bea.core.encrypted` package to the `Import-Package` header. See [Section 24.2.2.1, "Creating the MANIFEST.MF File."](#)
9. Re-assemble and deploy your application as usual. See [Chapter 24, "Assembling and Deploying Oracle CEP Applications."](#)

14.3.3 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.

The code snippets below build on the HelloWorld adapter Java code, shown in [Section 14.2.2.1, "Implementing a Custom Adapter as an Event Source."](#)

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 19.3, "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.

14.4 Configuring the Custom Adapter EPN Assembly File

The adapter and adapter factory (if used) must be registered in the EPN assembly file, as discussed in the following sections:

- [Section 14.4.1, "Registering the Custom Adapter Factory"](#)
- [Section 14.4.2, "Declaring the Custom Adapter Components in your Application"](#)

For a complete description of the configuration file, including registration of other components of your application, see [Section 4.3, "Creating EPN Assembly Files."](#)

14.4.1 Registering the Custom Adapter Factory

You register factories in the EPN assembly file using the `wlevs:factory` element:

```
<wlevs:factory provider-name="myprovider" class="my.Implementation"/>
```

If you need to specify service properties, then you must use the `osgi:service` element to register the factory as an OSGI service in the EPN assembly file. The scope of the OSGI service registry is the entire Oracle CEP. This means that if more than one application deployed to a given server is going to use the same adapter factory, be sure to register the adapter factory only *once* as an OSGI service.

Add an entry to register the service as an implementation of the `com.bea.wlevs.ede.api.AdapterFactory` interface. Provide a property, with the key attribute equal to `type`, and the name by which this adapter provider will be referenced. Finally, add a nested standard Spring bean element to register your specific adapter class in the Spring application context

For example, the following segment of the EPN assembly file registers the HelloWorldAdapterFactory as the provider for type hellomsgs:

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

For more information on how to reference a factory by its type, see [Section 14.4.2, "Declaring the Custom Adapter Components in your Application"](#).

14.4.2 Declaring the Custom Adapter Components in your Application

In the EPN assembly file, you use the `wlevs:adapter` element to declare an adapter as a component in the event processor network. For example:

```
<wlevs:adapter id="recplayEventSink"
  class="com.bea.wlevs.example.recplayRecplayEventSink">
  <wlevs:listener ref="playbackHttpPublisher"/>
</wlevs:adapter>
```

If you registered an optional factory as an OSGI service, then use the `provider` attribute to point to the name you specified as the `type` in your `osgi:service` entry; for example:

```
<wlevs:adapter id="helloworldAdapter" provider="hellomsgs"/>
```

This means that an adapter will be instantiated by the factory registered for the type `hellomsgs`.

You can also use a `wlevs:instance-property` child element of `wlevs:adapter` to set any *static* properties in the adapter bean. Static properties are those that you will not dynamically change after the adapter is deployed.

For example, if your adapter class has a `setPort` method, you can pass it the port number as shown:

```
<wlevs:adapter id="myAdapter" provider="myProvider">
  <wlevs:instance-property name="port" value="9001" />
</wlevs:adapter>
```

14.5 Configuring the Custom Adapter Component Configuration File

Each adapter in your application has a default configuration, and, optionally, an extended component configuration.

If your application has more than one adapter, you can create separate component configuration XML files for each adapter, or create a single component configuration XML file that contains the configuration for all adapters, or even all components of your application (adapters, processors, and streams). Choose the method that best suits your development environment.

The following procedure describes the main steps to create the adapter configuration file. For simplicity, it is assumed in the procedure that you are going to configure all components of an application in a single XML file

For more information, see:

- [Section 19, "Extending Component Configuration"](#)
- [Section B.2, "Component Configuration Schema wlevs_application_config.xsd"](#)

14.5.1 How to Configure a Custom Adapter Manually

The following procedure describes how to configure a custom adapter manually.

To configure the custom adapter component configuration file:

1. Create an XML file using your favorite XML editor.

You can name this XML file anything you want, provided it ends with the `.xml` extension.

The root element of the configuration file is `config`, with namespace definitions shown in the next step.

2. For each adapter in your application, add an `adapter` child element of `config`.

Uniquely identify each adapter with the `name` child element. This name must be the same as the value of the `id` attribute in the `wlevs:adapter` element of the EPN assembly file that defines the event processing network of your application. This is how Oracle CEP knows to which particular adapter component in the EPN assembly file this adapter configuration applies. See [Section 4.3, "Creating EPN Assembly Files"](#) for details.

For example, if your application has two adapters, the configuration file might initially look like this:

```
<?xml version="1.0" encoding="UTF-8"?>
<helloworld:config
  xmlns:helloworld="http://www.bea.com/xml/ns/wlevs/example/helloworld">
  <processor>
    ...
  </processor>
  <adapter>
    <name>firstAdapter</name>
    ...
  </adapter>
  <adapter>
    <name>secondAdapter</name>
    ...
  </adapter>
</helloworld:config>
```

In the example, the configuration file includes two adapters called `firstAdapter` and `secondAdapter`. This means that the EPN assembly file must include at least two adapter registrations with the same identifiers:

```
<wlevs:adapter id="firstAdapter" ...>
  ...
</wlevs:adapter>
<wlevs:adapter id="secondAdapter" ...>
  ...
</wlevs:adapter>
```

Caution: Identifiers and names in XML files are case sensitive, so be sure you specify the same case when referencing the component's identifier in the EPN assembly file.

14.5.1.1 Example of a Custom Adapter Configuration File

The following sample XML file shows how to configure two adapters, `firstAdapter` and `secondAdapter`.

```
<?xml version="1.0" encoding="UTF-8"?>
<sample:config
  xmlns:sample="http://www.bea.com/xml/ns/wlavs/example/sample">
  <adapter>
    <name>firstAdapter</name>
  </adapter>
  <adapter>
    <name>secondAdapter</name>
  </adapter>
</sample:config>
```

Configuring Custom Event Beans

This chapter describes how to code and register custom event beans, including:

- [Section 15.1, "Overview of Custom Event Beans"](#)
- [Section 15.2, "Implementing a Custom Event Bean"](#)
- [Section 15.3, "Configuring the Custom Event Bean EPN Assembly File"](#)
- [Section 15.4, "Configuring the Custom Event Bean Component Configuration File"](#)

15.1 Overview of Custom Event Beans

An event bean is a Plain Old Java Object (POJO) managed by the Oracle CEP management framework. You register event beans in the EPN assembly file using the Oracle CEP `wlevs:event-bean` stage rather than the standard bean element.

An event bean is a type of Stage, it can be monitored by the Oracle CEP monitoring framework, make use of the configuration metadata annotations, and it can be set to record, and play-back events that pass through it. An event bean can also participate in the Oracle CEP server bean lifecycle by specifying methods in its XML declaration, rather than by implementing Oracle CEP server API interfaces.

To add a Plain Old Java Object (POJO) to your Oracle CEP application:

- Use an event bean to actively use the capabilities of the Oracle CEP server container.
- Use a Spring bean for legacy integration to Spring.

For more information, see [Chapter 16, "Configuring Custom Spring Beans"](#).

15.1.1 Custom Event Bean Event Sources and Event Sinks

Event beans can be event sources, event sinks, or both. Event sources generate events, event sinks receive events.

15.1.1.1 Custom Event Beans as Event Sources

You specify that an event bean component in your EPN is an event source by implementing the `com.bea.wlevs.ede.api.StreamSource` or `RelationSource` API.

The implementation class of an event bean that is an event source may be specified as private to the application or as an advertised OSGI service re-usable by other applications.

For the framework to be able to fully manage the custom event bean as an EPN component, it must be specified as an event bean rather than a standard Spring bean. Management tasks include monitoring and record/playback.

You register event beans in the EPN assembly file using the `wlevs:event-bean` element. For example:

```
<wlevs:event-bean id="myEventBeanSource" class="com.acme.MySourceEventBean">
</wlevs:event-bean>
```

In this example, the Java class `MySourceEventBean.java` implements the `com.bea.wlevs.ede.api.StreamSource` or `RelationSource` API.

For more information, see [Section 15.2.1, "Implementing a Custom Event Bean as an Event Source"](#).

15.1.1.2 Custom Event Beans as Event Sinks

The functionality of custom event beans as event sinks is very similar to that of event sources except that custom event bean sinks must implement the `com.bea.wlevs.ede.api.StreamSink` or `RelationSink` API.

Event sinks are not active which means that if they implement the `Runnable` interface, Oracle CEP does not run them in a separate thread.

You reference event sinks in the EPN assembly file using the `wlevs:listener` element:

```
<wlevs:event-bean id="myEventBeanSink" class="com.acme.MySinkEventBean">
</wlevs:event-bean>
...
<wlevs:channel id="myStream" >
  <wlevs:listener ref="myEventBeanSink" />
</wlevs:channel>
```

In this example, the Java class `MySinkEventBean.java` implements the `com.bea.wlevs.ede.api.StreamSink` or `RelationSink` API.

For more information, see [Section 15.2.2, "Implementing a Custom Event Bean as an Event Sink"](#)

15.1.2 Custom Event Bean Factories

If your event bean is going to be used only by a single Oracle CEP application, then you do not need to create a factory. However, if multiple applications are going to use the same event bean, then you should also program a factory. In this case, every application gets its own instance of the adapter.

Event bean factories must implement the `com.bea.wlevs.ede.api.Factory` interface. This interface has a single method, `create`, that you implement to create an adapter or event bean instance.

You register factories in the EPN assembly file using the `wlevs:factory` element:

```
<wlevs:factory provider-name="myprovider" class="my.Implementation"/>
```

Note that if you need to specify service properties, then you must use the `<osgi:service>` to register the factory.

15.2 Implementing a Custom Event Bean

The following procedure describes the typical steps for creating a custom event bean.

To implement a custom event bean:

Note: The following 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 24.2.4.2, "How to Assemble a Custom Event Bean in its Own Bundle."](#)

1. Program the custom event bean Java class.

If your custom event bean is an event source, see [Section 15.2.1, "Implementing a Custom Event Bean as an Event Source."](#)

If your custom event bean is an event sink, see [Section 15.2.2, "Implementing a Custom Event Bean as an Event Sink."](#)

If your custom event bean is both an event source and event sink, then see both sections.

2. Optionally program the factory class.

You only need to do this if many applications are going to use the custom event bean.

See [Section 15.2.3, "Implementing a Custom Event Bean Factory."](#)

3. Optionally extend the configuration of the custom event bean if its basic one is not adequate.

See [Chapter 19, "Extending Component Configuration."](#)

4. Configure the custom event bean.

For more information, see:

- [Section 15.3, "Configuring the Custom Event Bean EPN Assembly File."](#)
- [Section 15.4, "Configuring the Custom Event Bean Component Configuration File."](#)

15.2.1 Implementing a Custom Event Bean as an Event Source

The following example shows a custom event bean class as an event source ; see the explanation after the example for coding guidelines that correspond to the Java code in bold.

```
package com.acme;

import com.bea.wlevs.ede.api.StreamSender;
import com.bea.wlevs.ede.api.StreamSource;
import com.bea.wlevs.ede.api RunnableBean;

public class EventBeanSource implements RunnableBean, StreamSource {

    public void setEventSender (StreamSender streamSender) {
        ...
    }

    public void run() {
        ...
    }
}
```

```

    public synchronized void suspend() throws Exception {
        ...
    }
}

```

Follow these guidelines when programming the custom event bean Java class; code snippets of the guidelines are shown in bold in the preceding example:

- Import the interfaces and classes of the Oracle CEP API:

```

import com.bea.wlevs.ede.api.StreamSender;
import com.bea.wlevs.ede.api.StreamSource;
import com.bea.wlevs.ede.api.RunnableBean;

```

Because the custom event bean is an event source it must implement the `StreamSource` interface. If you want the custom event bean to run in a thread, also implement `RunnableBean`. The `StreamSender` interface sends event types to the next component in your application network. For full details of these APIs, see *Oracle Fusion Middleware Java API Reference for Oracle Complex Event Processing*.

- The custom event bean class must implement the `StreamSource` and `RunnableBean` interfaces because it is an event source and will run in its own thread:

```

public class HelloWorldAdapter implements RunnableBean, StreamSource {

```

The `StreamSource` interface provides the `StreamSender` that you use to send events.

- Because the custom event bean implements the `RunnableBean` interface, your adapter must then implement the `run` method:

```

public void run() { ...

```

This is where you should put the code that reads the incoming data, such as from a market feed, and convert it into an Oracle CEP event type, and then send the event to the next component in the network. Refer to the documentation of your data feed provider for details on how to read the incoming data. See [Section 24.2.2.2, "Accessing Third-Party JAR Files"](#) for information about ensuring you can access the vendor APIs if they are packaged in a third-party JAR file.

- Because the custom event bean implements `StreamSource`, you must implement the `setEventSender` method, which passes in the `StreamSender` that you use to send events:

```

public void setEventSender(StreamSender sender) { ...

```

- If, as is typically the case, your custom event bean implements `SuspendableBean`, you must implement the `suspend` method that stops the custom event bean when, for example, the application is undeployed:

```

public synchronized void suspend() throws Exception { ...

```

15.2.2 Implementing a Custom Event Bean as an Event Sink

The following sample code shows a Spring bean from HelloWorld application that acts as an event sink; see the explanation after the example for the code shown in bold:

```

package com.bea.wlevs.example.helloworld;

import com.bea.wlevs.ede.api.StreamSink;

```

```
import com.bea.wlevs.event.example.helloworld.HelloWorldEvent;

public class HelloWorldBean implements StreamSink {

    public void onInsertEvent(Object event) {
        if (event instanceof HelloWorldEvent) {
            HelloWorldEvent helloWorldEvent = (HelloWorldEvent) event;
            System.out.println("Message: " + helloWorldEvent.getMessage());
        }
    }
}
```

The programming guidelines shown in the preceding example are as follows:

- Your bean must import the event type of the application, which in the HelloWorld case is HelloWorldEvent:

```
import com.bea.wlevs.event.example.helloworld.HelloWorldEvent;
```

- Your bean must implement the com.bea.wlevs.ede.api.StreamSink interface:

```
public class HelloWorldBean implements StreamSink {...
```

- The StreamSink interface has a single method that you must implement, onInsertEvent(java.lang.Object), which is a callback method for receiving events. The parameter of the method is an Object that represents the actual event that the bean received from the component that sent it the event:

```
public void onInsertEvent(Object event)
```

- The data type of the events is determined by the event type you registered in the EPN assembly file of the application. In the example, the event type is HelloWorldEvent; the code first ensures that the received event is truly a HelloWorldEvent:

```
if (event instanceof HelloWorldEvent) {
    HelloWorldEvent helloWorldEvent = (HelloWorldEvent) event;
```

This event type is a JavaBean that was configured in the EPN assembly file as shown:

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

See [Section 4.3, "Creating EPN Assembly Files"](#) for procedural information about creating the EPN assembly file, and [Appendix C, "Schema Reference: EPN Assembly spring-wlevs-v11_1_1_3.xsd"](#) for reference information.

- Events are instances of the appropriate JavaBean, so you access the individual properties using the standard getXXX methods. In the example, the HelloWorldEvent has a property called message. You access this property using method getMessage:

```
System.out.println("Message: " + helloWorldEvent.getMessage());
```

For complete API reference information about the Oracle CEP APIs described in this section, see the *Oracle Fusion Middleware Java API Reference for Oracle Complex Event Processing*.

15.2.3 Implementing a Custom Event Bean Factory

Your adapter factory class must implement the `com.bea.wlevs.ede.api.EventBeanFactory` interface, which has a single method, `create`, in which you code the creation of your specific adapter class. Event beans implement `Factory`.

The following is a possible adapter factory class for the HelloWorld example:

```
package com.acem;

import com.bea.wlevs.ede.api.EventBean;
import com.bea.wlevs.ede.api.EventBeanFactory;

public class MyEventBeanFactory implements Factory {
    public MyEventBeanFactory() {
    }
    public synchronized EventBean create() throws IllegalArgumentException {
        return new MyEventBeanFactory();
    }
}
```

For full details of these APIs, see the *Oracle Fusion Middleware Java API Reference for Oracle Complex Event Processing*.

15.3 Configuring the Custom Event Bean EPN Assembly File

The custom event bean and custom event bean factory (if used) must be registered in the EPN assembly file, as discussed in the following sections:

- [Section 15.3.1, "Registering the Custom Event Bean Factory"](#)
- [Section 15.3.2, "Declaring the Custom Event Bean Components in your Application"](#)

For a complete description of the configuration file, including registration of other components of your application, see [Section 4.3, "Creating EPN Assembly Files."](#)

15.3.1 Registering the Custom Event Bean Factory

You register factories in the EPN assembly file using the `wlevs:factory` element:

```
<wlevs:factory provider-name="myprovider" class="my.Implementation"/>
```

If you need to specify service properties, then you must use the `osgi:service` element to register the factory as an OSGI service in the EPN assembly file. The scope of the OSGI service registry is the entire Oracle CEP. This means that if more than one application deployed to a given server is going to use the same adapter factory, be sure to register the adapter factory only *once* as an OSGI service.

Add an entry to register the service as an implementation of the `com.bea.wlevs.ede.api.EventBeanFactory` 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

For example, the following segment of the EPN assembly file registers the `MyEventBeanFactory` as the provider for type `hellomsgs`:

```
<osgi:service interface="com.bea.wlevs.ede.api.EventBeanFactory">
  <osgi:service-properties>
    <entry key="type" value="myprovider" />
  </osgi:service-properties>
  <bean class="com.acme.MyEventBeanFactory" />
</osgi:service>
```

For more information on how to reference a factory by its `type`, see [Section 15.3.2, "Declaring the Custom Event Bean Components in your Application"](#).

15.3.2 Declaring the Custom Event Bean Components in your Application

In the EPN assembly file, you use the `wlevs:event-bean` element to declare a custom event bean as a component in the event processor network. For example:

```
<wlevs:event-bean id="replayEventSink"
  class="com.bea.wlevs.example.replayReplayEventSink">
  <wlevs:listener ref="playbackHttpPublisher" />
</wlevs:event-bean>
```

If you registered an optional factory as an OSGI service, then use the `provider` attribute to point to the name you specified as the `type` in your `osgi:service` entry; for example:

```
<wlevs:event-bean id="myEventBean" provider="myprovider" />
```

This means that an adapter will be instantiated by the factory registered for the type `myprovider`.

You can also use a `wlevs:instance-property` child element of `wlevs:adapter` to set any *static* properties in the adapter bean. Static properties are those that you will not dynamically change after the adapter is deployed.

For example, if your adapter class has a `setPort` method, you can pass it the port number as shown:

```
<wlevs:event-bean id="myEventBean" provider="myProvider">
  <wlevs:instance-property name="port" value="9001" />
</wlevs:event-bean>
```

15.4 Configuring the Custom Event Bean Component Configuration File

Each custom event bean in your application has a default configuration, and, optionally, an extended component configuration.

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 (adapters, processors, and streams). Choose the method that best suits your development environment.

The following procedure describes the main steps to create the custom event bean configuration file. For simplicity, it is assumed in the procedure that you are going to configure all components of an application in a single XML file

For more information, see:

- [Section 19, "Extending Component Configuration"](#)
- [Section B.2, "Component Configuration Schema `wlevs_application_config.xsd`"](#)

15.4.1 How to Configure a Custom Event Bean Manually

The following procedure describes how to configure a custom event bean manually.

To configure the custom event bean component configuration file:

1. Create an XML file using your favorite XML editor.

You can name this XML file anything you want, provided it ends with the `.xml` extension.

The root element of the configuration file is `config`, with namespace definitions shown in the next step.

2. For each event bean in your application, add an `event-bean` child element of `config`.

Uniquely identify each custom 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 CEP knows to which particular custom event bean component in the EPN assembly file this adapter configuration applies. See [Section 4.3, "Creating EPN Assembly Files"](#) for details.

For example, if your application has two custom event beans, the configuration file might initially look like this:

```
<?xml version="1.0" encoding="UTF-8"?>
<helloworld:config
  xmlns:helloworld="http://www.bea.com/xml/ns/wlevs/example/helloworld">
  <processor>
    ...
  </processor>
  <event-bean>
    <name>firstEventBean</name>
    ...
  </event-bean>
  <event-bean>
    <name>firstEventBean</name>
    ...
  </event-bean>
</helloworld:config>
```

In the example, the configuration file includes two custom event beans called `firstEventBean` and `secondEventBean`. This means that the EPN assembly file must include at least two custom event bean registrations with the same identifiers:

```
<wlevs:event-bean id="firstEventBean" ...>
  ...
</wlevs:event-bean>
<wlevs:event-bean id="secondEventBean" ...>
  ...
</wlevs:event-bean>
```

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.

15.4.1.1 Example of a Custom Event Bean Configuration File

The following sample XML file shows how to configure two custom event beans, `firstEventBean` and `secondEventBean`.

```
<?xml version="1.0" encoding="UTF-8"?>
<sample:config
  xmlns:sample="http://www.bea.com/xml/ns/wlevs/example/sample">
  <event-bean>
    <name>firstEventBean</name>
    ...
  </event-bean>
  <event-bean>
    <name>firstEventBean</name>
    ...
  </event-bean>
</sample:config>
```

Configuring Custom Spring Beans

This chapter describes how to code and register custom Spring beans, including:

- [Section 16.1, "Overview of Custom Spring Beans"](#)
- [Section 16.2, "Implementing a Custom Spring Bean"](#)
- [Section 16.3, "Configuring the Custom Spring Bean EPN File"](#)

16.1 Overview of Custom Spring Beans

A Spring bean is a Plain Old Java Objects (POJO) managed by the Spring framework. You register a Spring bean in the EPN assembly file using the standard `bean` element.

A Spring bean is not a type of stage: it cannot be monitored by the Oracle CEP monitoring framework, cannot use the configuration metadata annotations, and cannot be set to record and play-back events that pass through it.

To add a POJO to your Oracle CEP application:

- Use a Spring bean for legacy integration to Spring.
- Use an event bean to actively use the capabilities of the Oracle CEP server container.

For more information, see [Chapter 15, "Configuring Custom Event Beans"](#).

16.1.1 Spring Bean Event Sources and Event Sinks

Standard Spring beans can be event sources, event sinks, or both. Event sources generate events, event sinks receive events.

16.1.1.1 Spring Beans as Event Sources

You specify that a standard Spring bean component in your EPN is an event source by implementing the `com.bea.wlevs.ede.api.StreamSource` or `RelationSource` API.

The bean may also optionally implement the various lifecycle interfaces, such as `InitializingBean`, `DisposableBean`, and the active interfaces, such as `RunnableBean`. If a Spring-bean implements `Runnable` but not `RunnableBean`, Oracle CEP does not run it in a thread. This is different behavior from an event bean.

The Spring bean event source can make use of the configuration metadata annotations, such as `@Prepare`, `@Rollback`, and `@Activate`.

You register the Spring bean in the EPN assembly file in the standard way using the `bean` element. For example:

```
<bean id="bean" class="com.acme.BeanSource" />
```

In this example, the Java class `BeanSource.java` implements the `com.bea.wlevs.ede.api.StreamSource` or `RelationSource` API.

16.1.1.2 Spring Beans as Event Sinks

You specify that a standard Spring bean component in your EPN is an event sink by implementing the `com.bea.wlevs.ede.api.StreamSink` or `RelationSink` API.

The bean may also optionally implement the various lifecycle interfaces, such as `InitializingBean`, `DisposableBean`, and the active interfaces, such as `RunnableBean`. If a Spring-bean implements `Runnable` but not `RunnableBean`, Oracle CEP does not run it in a thread. This is different behavior from an event bean.

The Spring bean event source can make use of the configuration metadata annotations, such as `@Prepare`, `@Rollback`, and `@Activate`.

You register the Spring bean in the EPN assembly file in the standard way using the `bean` element. You can then specify this bean as an event sink of some other stage in the EPN.

You reference event sinks in the EPN assembly file using the `wlevs:listener` element:

```
<wlevs:channel id="S2" advertise="true" event-type="StockEvent" >
  <wlevs:listener ref="bean"/>
  <wlevs:source ref="cacheProcessor"/>
</wlevs:channel>
```

```
<bean id="bean" class="com.bea.wlevs.example.cachecql.Bean" />
```

In this example, the Java class `Bean.java` implements the `com.bea.wlevs.ede.api.StreamSink` or `RelationSink` API.

16.2 Implementing a Custom Spring Bean

The following procedure describes the typical steps for creating a custom event bean.

To implement a custom event bean:

1. Program the custom event bean Java class.

If your custom event bean is an event source, see [Section 16.2.1, "Implementing a Custom Spring Bean as an Event Source."](#)

If your custom event bean is an event sink, see [Section 16.2.2, "Implementing a Custom Spring Bean as an Event Sink."](#)

If your custom event bean is both an event source and event sink, then see both sections.

2. Update the EPN assembly file with custom event bean and custom event bean factory (if used) registration info.

See [Section 16.3, "Configuring the Custom Spring Bean EPN File."](#)

3. Optionally extend the configuration of the custom event bean if its basic one is not adequate.

See [Chapter 19, "Extending Component Configuration."](#)

16.2.1 Implementing a Custom Spring Bean as an Event Source

The following example shows a custom Spring bean class as an event source ; see the explanation after the example for coding guidelines that correspond to the Java code in bold.

```
package com.acme;

import com.bea.wlevs.ede.api.StreamSender;
import com.bea.wlevs.ede.api.StreamSource;
import com.bea.wlevs.ede.api.RunnableBean;

public class SpringBeanSource implements RunnableBean, StreamSource {

    public void setEventSender (StreamSender streamSender) {
        ...
    }

    public void run() {
        ...
    }

    public synchronized void suspend() throws Exception {
        ...
    }
}
```

Follow these guidelines when programming the custom Spring bean Java class; code snippets of the guidelines are shown in bold in the preceding example:

- Import the interfaces and classes of the Oracle CEP API:

```
import com.bea.wlevs.ede.api.StreamSender;
import com.bea.wlevs.ede.api.StreamSource;
import com.bea.wlevs.ede.api.RunnableBean;
```

Because the custom Spring bean is an event source it must implement the `StreamSource` interface. If you want the custom Spring bean to run in a thread, also implement `RunnableBean`. The `StreamSender` interface sends event types to the next component in your application network. For full details of these APIs, see *Oracle Fusion Middleware Java API Reference for Oracle Complex Event Processing*.

- The custom Spring bean class must implement the `StreamSource` and `RunnableBean` interfaces because it is an event source and will run in its own thread:

```
public class HelloWorldAdapter implements RunnableBean, StreamSource {
```

The `StreamSource` interface provides the `StreamSender` that you use to send events.

- Because the custom Spring bean implements the `RunnableBean` interface, your adapter must then implement the `run` method:

```
public void run() {...
```

This is where you should put the code that reads the incoming data, such as from a market feed, and convert it into an Oracle CEP event type, and then send the event to the next component in the network. Refer to the documentation of your data feed provider for details on how to read the incoming data. See [Section 24.2.2.2, "Accessing Third-Party JAR Files"](#) for information about ensuring you can access the vendor APIs if they are packaged in a third-party JAR file.

- Because the custom Spring bean implements `StreamSource`, you must implement the `setEventSender` method, which passes in the `StreamSender` that you use to send events:

```
public void setEventSender(StreamSender sender) { ...
```

- If, as is typically the case, your custom Spring bean implements `SuspendableBean`, you must implement the `suspend` method that stops the adapter when, for example, the application is undeployed:

```
public synchronized void suspend() throws Exception { ...
```

16.2.2 Implementing a Custom Spring Bean as an Event Sink

The following sample code shows a Spring bean from HelloWorld application that acts as an event sink; see the explanation after the example for the code shown in bold:

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

import com.bea.wlevs.ede.api.StreamSink;
import com.bea.wlevs.event.example.helloworld.HelloWorldEvent;

public class HelloWorldBean implements StreamSink {

    public void onInsertEvent(Object event) {
        if (event instanceof HelloWorldEvent) {
            HelloWorldEvent helloWorldEvent = (HelloWorldEvent) event;
            System.out.println("Message: " + helloWorldEvent.getMessage());
        }
    }
}
```

The programming guidelines shown in the preceding example are as follows:

- Your bean must import the event type of the application, which in the HelloWorld case is `HelloWorldEvent`:

```
import com.bea.wlevs.event.example.helloworld.HelloWorldEvent;
```

- Your bean must implement the `com.bea.wlevs.ede.api.StreamSink` interface:

```
public class HelloWorldBean implements StreamSink {...
```

- The `StreamSink` interface has a single method that you must implement, `onInsertEvent(java.lang.Object)`, which is a callback method for receiving events. The parameter of the method is an `Object` that represents the actual event that the bean received from the component that sent it the event:

```
public void onInsertEvent(Object event)
```

- The data type of the events is determined by the event type you registered in the EPN assembly file of the application. In the example, the event type is `HelloWorldEvent`; the code first ensures that the received event is truly a `HelloWorldEvent`:

```
if (event instanceof HelloWorldEvent) {
    HelloWorldEvent helloWorldEvent = (HelloWorldEvent) event;
```

This event type is a `JavaBean` that was configured in the EPN assembly file as shown:

```

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

```

See [Section 4.3, "Creating EPN Assembly Files"](#) for procedural information about creating the EPN assembly file, and [Appendix C, "Schema Reference: EPN Assembly spring-wlevs-v11_1_1_3.xsd"](#) for reference information.

- Events are instances of the appropriate JavaBean, so you access the individual properties using the standard `getXXX` methods. In the example, the `HelloWorldEvent` has a property called `message`. You access this property using method `getMessage`:

```
System.out.println("Message: " + helloWorldEvent.getMessage());
```

For complete API reference information about the Oracle CEP APIs described in this section, see the *Oracle Fusion Middleware Java API Reference for Oracle Complex Event Processing*.

16.3 Configuring the Custom Spring Bean EPN File

The custom event bean and custom event bean factory (if used) must be registered in the EPN assembly file, as discussed in the following sections:

- [Section 16.3.1, "Declaring the Custom Spring Bean Components in your Application"](#)

For a complete description of the configuration file, including registration of other components of your application, see [Section 4.3, "Creating EPN Assembly Files."](#)

16.3.1 Declaring the Custom Spring Bean Components in your Application

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="replayEventSink"
      class="com.bea.wlevs.example.replayReplayEventSink">
</bean>

```

Configuring Web Services

You can use Web Services from within an Oracle CEP application.

This chapter describes:

- [Section 17.1, "Understanding Oracle CEP and Web Services"](#)
- [Section 17.2, "How to Invoke a Web Service From an Oracle CEP Application"](#)
- [Section 17.3, "How to Expose an Oracle CEP Application as a Web Service"](#)

17.1 Understanding Oracle CEP and Web Services

You can integrate an Oracle CEP application with other systems using Web Services.

Oracle CEP 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)

17.2 How to Invoke a Web Service From an Oracle CEP Application

This procedure describes how to create an Oracle CEP application that invokes a Web Service. In this scenario, the Oracle CEP application is the Web Service client.

To invoke a Web Service from an Oracle CEP 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 CEP (such as `/oracle_home`).

3. Archive the generated `.class` files within the Oracle CEP application JAR file.

For more information, see [Section 4.7, "Managing Libraries and Other Non-Class Files in Oracle CEP 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 4.7.4, "How to Export a Package"](#).

5. Import the following packages to the Oracle CEP 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 4.7, "Managing Libraries and Other Non-Class Files in Oracle CEP 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");
```

17.3 How to Expose an Oracle CEP Application as a Web Service

This procedure describes how to expose an Oracle CEP application as a Web Service. In this scenario, the Oracle CEP application is the Web Service provider.

To expose an Oracle CEP 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 17-1](#) shows. [Table 17-1](#) describes the attributes in this file.

Example 17–1 *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 17–1 *bea-jaxws.xml File Attributes*

Attribute	Description
name	The name of the web service.
implementation-class	The class that implements the service.
url-pattern	The url pattern to access the webservice.
wsdl-location	Relative path to the wsdl in the bundle.
service-name	QName of the service.
port-name	QName of the port.

For more information, see [Section 4.7, "Managing Libraries and Other Non-Class Files in Oracle CEP 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 4.7, "Managing Libraries and Other Non-Class Files in Oracle CEP Projects"](#).

5. Import the following packages to the Oracle CEP 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 4.7, "Managing Libraries and Other Non-Class Files in Oracle CEP Projects"](#).

6. Add a `glassfish-ws` element to the Oracle CEP server `DOMAIN_DIR/config/config.xml` file that describes your Oracle CEP domain, where `DOMAIN_DIR` refers to your domain directory:

```
<glassfish-ws>  
  <name>JAXWS</name>  
  <http-service-name>JettyServer</http-service-name>  
</glassfish-ws>
```

Configuring Applications With Data Cartridges

This section describes:

- [Section 18.1, "Understanding Data Cartridge Application Context"](#)
- [Section 18.2, "How to Configure Oracle Spatial Application Context"](#)
- [Section 18.3, "How to Configure Oracle JDBC Data Cartridge Application Context"](#)

For more information on data cartridges, see "Introduction to Data Cartridges" in the *Oracle Complex Event Processing CQL Language Reference*.

18.1 Understanding Data Cartridge Application Context

Depending on the data cartridge implementation, you may be able to define an application context that the Oracle CEP 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 18.2, "How to Configure Oracle Spatial Application Context"](#)
- [Section 18.3, "How to Configure Oracle JDBC Data Cartridge Application Context"](#)

For more information on data cartridges, see "Introduction to Data Cartridges" in the *Oracle Complex Event Processing CQL Language Reference*.

18.2 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 CEP 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
- "Understanding Oracle Spatial" in the *Oracle Complex Event Processing CQL Language Reference*

To configure Oracle Spatial application context:

1. Open the EPN editor in the Oracle CEP IDE for Eclipse.
See [Section 6.1, "Opening the EPN Editor"](#).
2. Import the package `com.oracle.cep.cartridge.spatial` into your Oracle CEP application's `MANIFEST.MF` file.
For more information, see [Section 4.7.5, "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_3.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 Complex Event Processing CQL Language Reference*.

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"> ▪ CARTESIAN: for cartesian coordinate system. ▪ LAT_LNG_WGS84_SRID: for WGS84 coordinate system. ▪ An integer value from the Oracle Spatial SDO_COORD_SYS table COORD_SYS_ID column. 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 Complex Event Processing CQL Language Reference*.

18.3 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 `jdbc: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 Complex Event Processing CQL Language Reference*.

To configure Oracle JDBC data cartridge application context:

1. Open the EPN editor in the Oracle CEP IDE for Eclipse.
See [Section 6.1, "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_3.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 Complex Event Processing CQL Language Reference*.

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 CEP server `config.xml` file.

For more information, see:

- "Configuring JDBC for Oracle CEP" in the *Oracle Complex Event Processing Administrator's Guide*.
- [Section F.10, "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 Complex Event Processing CQL Language Reference*.

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 Complex Event Processing CQL Language Reference*.

Extending Component Configuration

This chapter describes:

- [Section 19.1, "Overview of Extending Component Configuration"](#)
- [Section 19.2, "Extending Component Configuration"](#)
- [Section 19.3, "Programming Access to the Configuration of a Custom Adapter or Event Bean"](#)

19.1 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 CEP. By extending the XSD Schema, you can add as many new elements to the adapter configuration as you want, with few restrictions other than each new element must have a name attribute.

This feature is based on standard technologies, such as XSD 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 19.1.1, "Extending Component Configuration Using Annotations"](#).
- Manually generating an XSD.
See [Section 19.1.2, "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 14, "Configuring Custom Adapters"](#)
- [Chapter 15, "Configuring Custom Event Beans"](#)
- [Section B.2, "Component Configuration Schema `wlevs_application_config.xsd`"](#)

19.1.1 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 CEP 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 CEP 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 19.2.1, "How to Extend Component Configuration Using Annotations"](#)
- <http://java.sun.com/javase/6/docs/api/javax/xml/bind/annotation/package-summary.html>

19.1.2 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 19.2.2, "How to Extend Component Configuration Using an XSD"](#)
- <https://jaxb.dev.java.net/>

19.2 Extending Component Configuration

You can extend component configuration in either of the following ways:

For more information, see [Section 19.1, "Overview of Extending Component Configuration"](#).

19.2.1 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 19.2.2, "How to Extend Component Configuration Using an XSD"](#) describes.

For more information, see [Section 19.1.1, "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 14.2, "Implementing a Custom Adapter"](#)
 - [Section 15.2, "Implementing a Custom Event Bean"](#)
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 19.2.2, "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 19–1](#) 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 19–1 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 19.3, "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 14.5, "Configuring the Custom Adapter 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 19-2](#) shows a component configuration file for the custom adapter in [Example 19-1](#).

Example 19-2 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 19-5](#) shows.

6. Package and deploy your application.

For more information, see [Chapter 24, "Assembling and Deploying Oracle CEP Applications"](#).

19.2.2 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 19.2.1, "How to Extend Component Configuration Using Annotations"](#) describes.

For more information, see [Section 19.1.2, "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 CEP, to describe them.

See [Section 19.2.2.1, "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 CEP distribution for your convenience.

The following sample `build.xml` file shows how to do this:

```
<property name="base.dir" value="." />
<property name="output.dir" value="output" />
<property name="sharedlib.dir" value="${base.dir}/../../../../../../modules" />
<property name="wlrllib.dir" value="${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 CEP XSD files (`wlevs_base_config.xsd` and `wlevs_application_config.xsd`) to the same directory as the `helloworld.xsd` file because `helloworld.xsd` imports the Oracle CEP XSD files.

For more information, see

<https://jaxb.dev.java.net/jaxb20-ea/docs/xjcTask.html>.

3. Compile these generated Java files into classes.
4. Package the compiled Java class files in your application bundle.

See [Section 24.2, "Assembling an Oracle CEP Application"](#).

5. Program your custom adapter or event bean.

For more information, see:

- [Section 14.2, "Implementing a Custom Adapter."](#)
 - [Section 15.2, "Implementing a Custom Event Bean."](#)
6. Within your custom adapter or event bean code, access the extended configuration as [Section 19.3, "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 19–3](#) shows a component configuration file for the XSD you created in [Section 19.2.2.1, "Creating the XSD Schema File"](#).

Example 19–3 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>
```

19.2.2.1 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 19.2.2.1.1, "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 B.2, "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 10.1, "Overview of Oracle CQL Processor Configuration"](#)
 - [Section 11.1, "Overview of EPL Processor Component Configuration"](#)
- `DefaultStreamConfig`—For a description of the default channel configuration, see [Section 9.1, "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>
    
```

5. Optionally use the `jxb:package` child element of `jxb:schemaBindings` to specify the package name of the generated Java code:

```

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

19.2.2.1.1 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:schema>
    
```

19.3 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 CEP maps the configuration of each component (specified in the component configuration XML files) into Java objects using the Java Architecture for XML Binding (JAXB) standard (for more information, see <https://jaxb.dev.java.net/>). Because there is a single XML element that contains the configuration data for each component, JAXB in turn also produces a single Java class that represents this configuration data. Oracle CEP passes an instance of this Java class to the component (processor, channel, or adapter) at runtime when the component is initialized, and also whenever there is a dynamic change to the component's configuration.

You can access this component configuration at runtime in either of the following ways:

- Using resource injection as [Section 19.3.1, "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 19.3.2, "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.

19.3.1 How to Access Component Configuration Using Resource Injection

By default, Oracle CEP configures adapters by direct injection of their Java bean properties followed by the usual configuration callbacks.

Consider the annotated custom adapter implementation that [Example 19–4](#) shows.

Example 19–4 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 19–5](#)

Example 19–5 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 run time, by default Oracle CEP directly injects the value (`true`) of the Java bean property `doit`.

Note: The extended component configuration schema requires a nested `config` element as [Example 19–5](#) shows.

For more information, see:

- [Section 19.3.2, "How to Access Component Configuration Using Lifecycle Callbacks"](#)
- [Section 1.4, "Configuring Oracle CEP Resource Access"](#)

19.3.2 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 CEP at runtime.

Oracle CEP 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 19.3.2.1, "Lifecycle Callback Annotations"](#)
- [Section 19.3.2.2, "Lifecycle"](#)

19.3.2.1 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 I.2, "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 I.3, "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 I.4, "com.bea.wlevs.configuration.Rollback"](#) for additional details about using this annotation in your adapter code.

For more information, see [Section 19.3.2.2, "Lifecycle"](#).

19.3.2.2 Lifecycle

Oracle CEP 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 19.3.1, "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.

Part V

Developing Applications for High Availability

Part V contains the following chapters:

- [Chapter 20, "Understanding High Availability"](#)
- [Chapter 21, "Configuring High Availability"](#)

Understanding High Availability

This chapter describes Oracle CEP components and design patterns that you can use to increase the availability of your Oracle CEP applications, including:

- [Section 20.1, "High Availability Architecture"](#)
- [Section 20.2, "Choosing a Quality of Service"](#)
- [Section 20.3, "Designing an Oracle CEP Application for High Availability"](#)

For more information on how to implement a particular high availability quality of service, see [Chapter 21, "Configuring High Availability"](#).

20.1 High Availability Architecture

Like any computing resource, Oracle CEP servers can be subject to both hardware and software faults that can lead to data- or service-loss. Oracle CEP high availability options seek to mitigate both the likelihood and the impact of such faults.

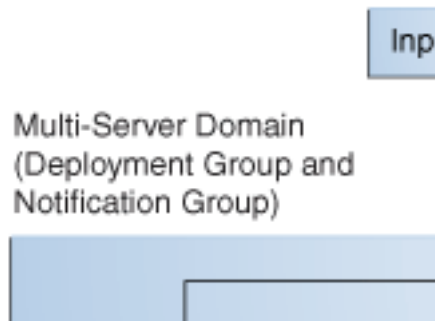
Oracle CEP supports an active-standby high availability architecture. This approach has the advantages of high performance, simplicity, and short failover time.

An Oracle CEP application that needs to be highly available is deployed to a group of two or more Oracle CEP server instances running in an Oracle CEP multi-server domain. Oracle CEP 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 CEP 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 20-1](#) shows a typical configuration.

Figure 20–1 Oracle CEP High Availability: Primary and Secondary Servers



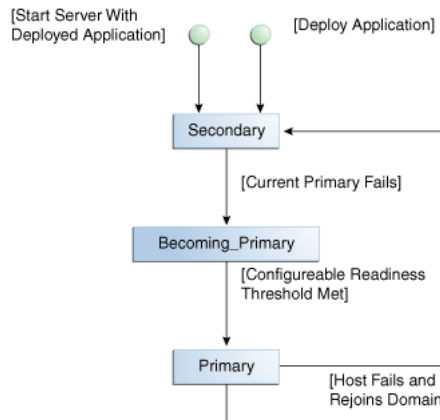
This section describes:

- [Section 20.1.1, "High Availability Lifecycle and Failover"](#)
- [Section 20.1.2, "Deployment Group and Notification Group"](#)
- [Section 20.1.3, "High Availability Components"](#)
- [Section 20.1.4, "High Availability and Scalability"](#)
- [Section 20.1.5, "High Availability and Oracle Coherence"](#)

20.1.1 High Availability Lifecycle and Failover

Figure 20–2 shows a state diagram for the Oracle CEP high availability lifecycle. In this diagram, the state names (`SECONDARY`, `BECOMING_PRIMARY`, and `PRIMARY`) correspond to the Oracle CEP high availability adapter `RuntimeMBean` method `getState` return values. These states are specific to Oracle CEP.

Figure 20–2 Oracle CEP 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 CEP high availability lifecycle in more detail, including:

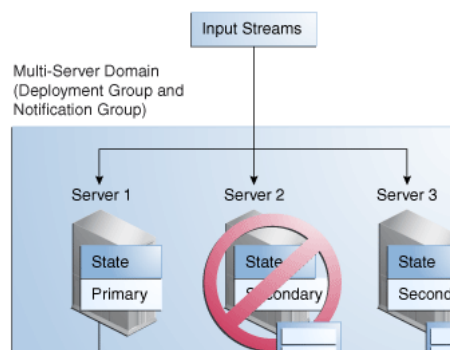
- [Section 20.1.1.1, "Secondary Failure"](#)

- [Section 20.1.1.2, "Primary Failure and Failover"](#)
- [Section 20.1.1.3, "Rejoining the High Availability Multi-Server Domain"](#)

20.1.1.1 Secondary Failure

In general, when a secondary server fails, there is no effect on Oracle CEP application operation as [Figure 20–3](#) shows. Regardless of the quality of service you choose, there are no missed or duplicate events.

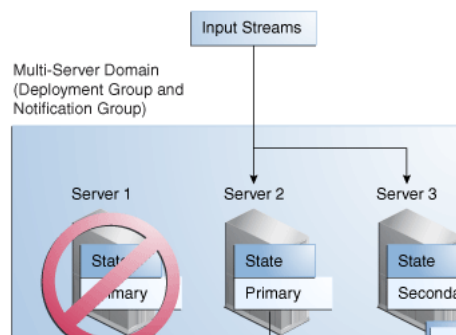
Figure 20–3 Secondary Failure



20.1.1.2 Primary Failure and Failover

However, when a primary server fails, as [Figure 20–4](#) shows, Oracle CEP performs a failover that may cause missed or duplicate events, depending on the quality of service you choose.

Figure 20–4 Primary Failure and Failover



During failover, Oracle CEP 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 20.2, "Choosing a Quality of Service"](#).

20.1.1.3 Rejoining the High Availability Multi-Server Domain

When a new Oracle CEP server is added to an Oracle CEP high availability multi-server domain or an existing failed server restarts, the server will not have fully joined the Oracle CEP 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` period). This `warm-up-window` time determines the minimum time it will take for the application to fully join the Oracle CEP 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` time and will fully join the Oracle CEP high availability deployment and notification groups as soon as it is deployed.

For more information, see [Section 20.3.2.5, "Choose an Adequate warm-up-window Time"](#).

20.1.2 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 CEP 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 CEP 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.

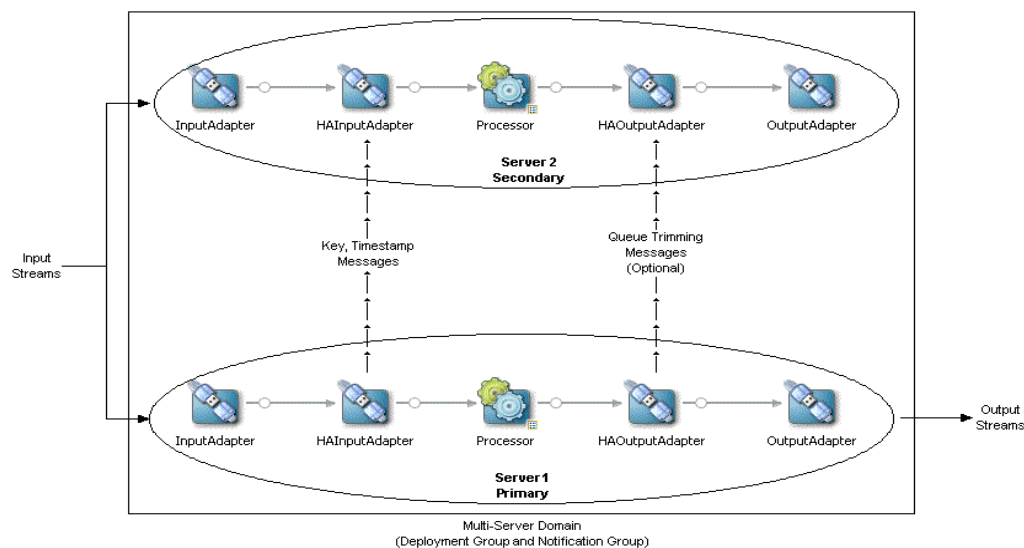
For more information, see:

- [Section 20.1.4, "High Availability and Scalability"](#)
- [Section 20.1.5, "High Availability and Oracle Coherence"](#)
- "How to Create an Oracle CEP Multi-Server Domain With Default Groups Using Oracle Coherence" in the *Oracle Complex Event Processing Administrator's Guide*.
- "How to Create an Oracle CEP Multi-Server Domain With Custom Groups Using Oracle Coherence" in the *Oracle Complex Event Processing Administrator's Guide*.

20.1.3 High Availability Components

To implement Oracle CEP 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 20-5](#) shows a typical EPN with all possible high availability adapters in place.

Figure 20–5 High Availability Adapters in the EPN

Note: For simplicity, Figure 20–5 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 CEP 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 CEP high availability provides one type of high availability input adapter. See [Section 20.1.3.1, "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 CEP 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 CEP high availability provides the following high availability output adapters:

- [Section 20.1.3.2, "Buffering Output Adapter"](#)
- [Section 20.1.3.3, "Broadcast Output Adapter"](#)
- [Section 20.1.3.4, "Correlating Output Adapter"](#)

Oracle CEP high availability also provides a notification groups Spring bean to increase scalability in JMS applications. See [Section 20.1.3.5, "ActiveActiveGroupBean"](#).

Which adapter you choose is determined by the high availability quality of service you choose. See [Section 20.2, "Choosing a Quality of Service"](#).

20.1.3.1 High Availability Input Adapter

The optional Oracle CEP high availability input adapter on the primary Oracle CEP 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 CEP high availability input adapter to send heartbeat events.

The Oracle CEP 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 20.2.1, "Simple Failover"](#) and [Section 20.2.2, "Simple Failover with Buffering"](#)). For these options, you should instead consider using application time with some incoming event property.

For more information, see:

- [Section 20.2.3, "Light-Weight Queue Trimming"](#)
- [Section 20.2.4, "Precise Recovery with JMS"](#)
- [Section 21.2.1, "How to Configure the High Availability Input Adapter"](#).

20.1.3.2 Buffering Output Adapter

The Oracle CEP 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 CEP 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 20.2.1, "Simple Failover"](#)
- [Section 20.2.2, "Simple Failover with Buffering"](#)
- [Section 21.2.2, "How to Configure the Buffering Output Adapter"](#).

20.1.3.3 Broadcast Output Adapter

The Oracle CEP 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 CEP high availability broadcast output adapter is applicable to the light-weight queue trimming high availability quality of service option.

For more information, see:

- [Section 20.2.3, "Light-Weight Queue Trimming"](#)

- [Section 21.2.3, "How to Configure the Broadcast Output Adapter"](#).

20.1.3.4 Correlating Output Adapter

The Oracle CEP 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 CEP high availability correlating output adapter is applicable to precise recovery with JMS high availability quality of service option.

For more information, see:

- [Section 20.2.4, "Precise Recovery with JMS"](#)
- [Section 21.2.4, "How to Configure the Correlating Output Adapter"](#).

20.1.3.5 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 CEP application scalability.

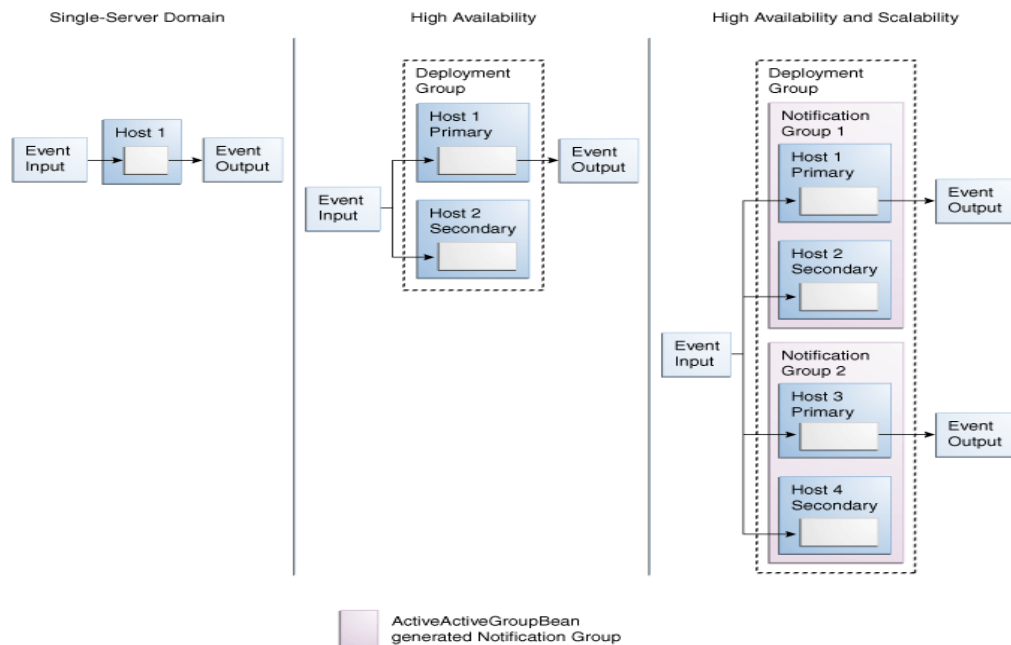
For more information, see:

- [Section 20.1.4, "High Availability and Scalability"](#)
- [Section 20.2.4, "Precise Recovery with JMS"](#)
- [Section 22.2.2, "ActiveActiveGroupBean"](#)

20.1.4 High Availability and Scalability

If you need to scale your Oracle CEP 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 20–6](#) shows three Oracle CEP application scenarios progressing from the simplest configuration, to high availability, and then to both high availability and scalability.

Figure 20–6 High Availability and Scalability

Most applications begin in a single-server domain without high availability. In this, the simplest scenario, an Oracle CEP application running on one Oracle CEP server processes an input event stream and produces output events.

In the high availability scenario, the Oracle CEP application has been configured to use Oracle CEP 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 CEP high availability application has been configured to use the `ActiveActiveGroupBean` to define notification groups. Each notification group contains two or more Oracle CEP 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 CEP 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 CEP high availability quality of service you configure.

For more information, see:

- [Section 22.2.2, "ActiveActiveGroupBean"](#)
- [Section 23.2.2, "How to Configure Scalability in a JMS Application With Oracle CEP High Availability"](#)

20.1.5 High Availability and Oracle Coherence

Oracle CEP high availability options depend on Oracle Coherence. You cannot implement Oracle CEP high availability options without Oracle Coherence.

When considering performance tuning, be sure to evaluate your Oracle Coherence configuration in addition to your Oracle CEP application.

For more information, see:

- [Section 27.2.5, "Oracle Coherence Performance Tuning Options"](#)
- "Configuring the Oracle Coherence Cluster" in the *Oracle Complex Event Processing Administrator's Guide*
- *Oracle Coherence Developer's Guide* at http://download.oracle.com/docs/cd/E15357_01/coh.360/e15723/toc.htm

20.2 Choosing a Quality of Service

Using Oracle CEP high availability, you may choose any of the quality of service options that [Table 20–1](#) 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 20–1 Oracle CEP High Availability Quality of Service

High Availability Option	Missed Events?	Duplicate Events?	Performance Overhead
Section 20.2.1, "Simple Failover"	Yes (many)	Yes (few)	Negligible
Section 20.2.2, "Simple Failover with Buffering"	Yes (few) ¹	Yes (many)	Low
Section 20.2.3, "Light-Weight Queue Trimming"	No	Yes (few)	Low-Medium ²
Section 20.2.4, "Precise Recovery with JMS"	No	No	High

¹ If you configure a big enough buffer then there will be no missed events.

² 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.

20.2.1 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 CEP 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 21.1.1, "How to Configure Simple Failover"](#).

20.2.2 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 CEP server rejoins the multi-server domain, if your application is an Oracle CEP 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 CEP 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 20.3.2.5, "Choose an Adequate warm-up-window Time"](#)
- [Section 21.1.2, "How to Configure Simple Failover With Buffering"](#)

20.2.3 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 CEP server rejoins the multi-server domain, if your application is an Oracle CEP 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 CEP 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 21.1.3, "How to Configure Light-Weight Queue Trimming"](#).

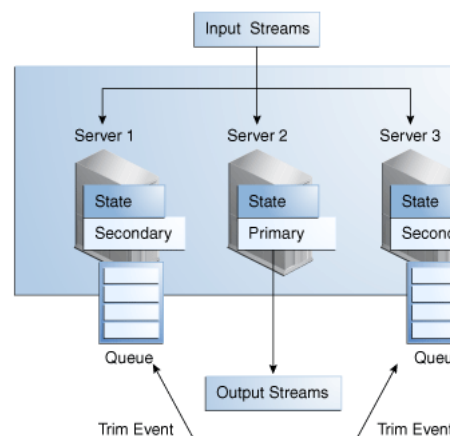
20.2.4 Precise Recovery with JMS

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 20–7](#) 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 20–7 *Precise Recovery with JMS*



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 no duplicate events are output.

When an Oracle CEP server rejoins the multi-server domain, if your application is an Oracle CEP 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 CEP 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 21.1.4, "How to Configure Precise Recovery With JMS"](#)
- [Section 23.2, "Configuring Scalability With the ActiveActiveGroupBean"](#)

20.3 Designing an Oracle CEP Application for High Availability

Although you can implement Oracle CEP high availability declaratively, to fully benefit from the high availability quality of service you choose, you must design your Oracle CEP application to take advantage of the high availability options that Oracle CEP provides.

When designing your Oracle CEP application for high availability, consider the following :

- [Section 20.3.1, "Primary Oracle CEP High Availability Use Case"](#)
- [Section 20.3.2, "High Availability Design Patterns"](#)
- [Section 20.3.3, "Oracle CQL Query Restrictions"](#)

20.3.1 Primary Oracle CEP High Availability Use Case

You can adapt Oracle CEP high availability options to various Oracle CEP application designs but in general, Oracle CEP 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 CEP high availability quality of service options are applicable:

- [Section 20.2.1, "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 CEP high availability quality of service options are applicable:
 - [Section 20.2.2, "Simple Failover with Buffering"](#)
 - [Section 20.2.3, "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 CEP high availability quality of service options are applicable
 - [Section 20.2.4, "Precise Recovery with JMS"](#)

20.3.2 High Availability Design Patterns

When designing your Oracle CEP application for use with Oracle CEP high availability options, observe the following design patterns:

- [Section 20.3.2.1, "Select the Minimum High Availability Your Application can Tolerate"](#)
- [Section 20.3.2.2, "Use Oracle CEP High Availability Components at All Ingress and Egress Points"](#)
- [Section 20.3.2.3, "Only Preserve What You Need"](#)
- [Section 20.3.2.4, "Limit Oracle CEP Application State"](#)
- [Section 20.3.2.5, "Choose an Adequate warm-up-window Time"](#)
- [Section 20.3.2.6, "Ensure Applications are Idempotent"](#)
- [Section 20.3.2.7, "Source Event Identity Externally"](#)
- [Section 20.3.2.8, "Understand the Importance of Event Ordering"](#)
- [Section 20.3.2.9, "Write Oracle CQL Queries with High Availability in Mind"](#)
- [Section 20.3.2.10, "Avoid Coupling Servers"](#)
- [Section 20.3.2.11, "Plan for Server Recovery"](#)

20.3.2.1 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.

20.3.2.2 Use Oracle CEP High Availability Components at All Ingress and Egress Points

You must use an Oracle CEP high availability input adapter after each regular input adapter and you must use an Oracle CEP high availability output adapter before each regular output adapter.

20.3.2.3 Only Preserve What You Need

Most Oracle CEP 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.

20.3.2.4 Limit Oracle CEP Application State

Oracle CEP 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.

20.3.2.5 Choose an Adequate warm-up-window Time

When a new Oracle CEP server is added to an Oracle CEP high availability multi-server domain or an existing failed server restarts, the server will not have fully joined the Oracle CEP 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 CEP high availability applications can be described as Type 1 or Type 2 applications as [Table 20–2](#) shows.

Table 20–2 Oracle CEP 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 20.1.1.3, "Rejoining the High Availability Multi-Server Domain"](#).

20.3.2.5.1 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 CEP 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 time*), after which it generates exactly the same stream of output events as other secondaries running the application.

The *warm-up-window time* is configured on an Oracle CEP high availability output adapter. The *warm-up-window 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 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 time*.

The Oracle CEP server uses system time during the *warm-up-window 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` 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` period. For example, an application that uses an tuple-based partitioned window that has no time qualification cannot use the `warm-up-window` period, since an arbitrary amount of time is required to rebuild the state of the window.

If a Type 1 application uses the Oracle CEP 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 20.3.3, "Oracle CQL Query Restrictions"](#)
- [Section 20.1.3.2, "Buffering Output Adapter"](#)
- [Section 20.1.3.3, "Broadcast Output Adapter"](#)
- [Section 20.1.3.4, "Correlating Output Adapter"](#)

20.3.2.5.2 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 CEP 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` 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 CEP 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.

20.3.2.6 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 CEP 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.

20.3.2.7 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 CEP system to provide this.

For more information, see [Section 20.3.3.6, "Prefer Application Time"](#).

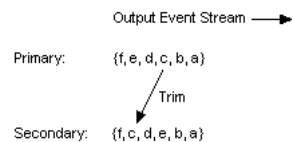
20.3.2.8 Understand the Importance of Event Ordering

For Oracle CEP 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 CEP 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 20–8](#). The primary has output events a, b, and c. After outputting event c, the primary sends the secondary a queue trimming message.

Figure 20–8 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 20.3.2.8.1, "Prefer Deterministic Behavior"](#)
- [Section 20.3.2.8.2, "Avoid Multithreading"](#)
- [Section 20.3.2.8.3, "Prefer Monotonic Event Identifiers"](#)

20.3.2.8.1 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.getTimeMillis` or `System.nanoTime` which can return different results on different machines because the system clocks are not synchronized.

20.3.2.8.2 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 CEP high availability adapter in parallel. If such a channel is an event source for an Oracle CEP 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 22.2.1, "EventPartitioner"](#)
- `max-threads` in [Table C-9, "Attributes of the `wlevs:channel` Application Assembly Element"](#)

20.3.2.8.3 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 CEP application using monotonic event identifiers. Using a monotonic event identifier, the Oracle CEP high availability adapter can handle an application that may produce events out of order.

20.3.2.9 Write Oracle CQL Queries with High Availability in Mind

Not all Oracle CQL query usage is supported when using Oracle CEP high availability. You may need to redefine your Oracle CQL queries to address these restrictions.

For more information, see [Section 20.3.3, "Oracle CQL Query Restrictions"](#).

20.3.2.10 Avoid Coupling Servers

The most performant high availability for Oracle CEP 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.

20.3.2.11 Plan for Server Recovery

When a secondary server rejoins the multi-server domain, the server must have time to rebuild the Oracle CEP application state to match that of the current primary and active secondaries as [Section 20.3.2.5, "Choose an Adequate warm-up-window 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.

20.3.3 Oracle CQL Query Restrictions

When writing Oracle CQL queries in an Oracle CEP application that uses Oracle CEP high availability options, observe the following restrictions:

- [Section 20.3.3.1, "Range-Based Windows"](#)
- [Section 20.3.3.2, "Tuple-Based Windows"](#)

- [Section 20.3.3.3, "Partitioned Windows"](#)
- [Section 20.3.3.4, "Sliding Windows"](#)
- [Section 20.3.3.5, "DURATION Clause and Non-Event Detection"](#)
- [Section 20.3.3.6, "Prefer Application Time"](#)

For more information on Oracle CQL, see the *Oracle Complex Event Processing CQL Language Reference*.

20.3.3.1 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 time. See also [Section 20.3.2.5, "Choose an Adequate warm-up-window Time"](#).

Channels must use application time if Oracle CQL queries contain range-based Windows. See also [Section 20.3.3.6, "Prefer Application Time"](#).

For more information, see "Range-Based Stream-to-Relation Window Operators" in the *Oracle Complex Event Processing CQL Language Reference*.

20.3.3.2 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 20.3.2.5, "Choose an Adequate warm-up-window Time"](#).

For more information, see "Tuple-Based Stream-to-Relation Window Operators" in the *Oracle Complex Event Processing CQL Language Reference*.

20.3.3.3 Partitioned Windows

Consider avoiding partitioned windows: there are cases where a partition cannot be rebuilt. If using partitioned windows, configure a warm-up-window time long enough to give the Oracle CEP server time to rebuild the partition. See also [Section 20.3.2.5, "Choose an Adequate warm-up-window Time"](#).

For more information, see "Partitioned Stream-to-Relation Window Operators" in the *Oracle Complex Event Processing CQL Language Reference*.

20.3.3.4 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 20.1.1.3, "Rejoining the High Availability Multi-Server Domain"](#)
- "S[range T1 slide T2]" in the *Oracle Complex Event Processing CQL Language Reference*
- "S [rows N1 slide N2]" in the *Oracle Complex Event Processing CQL Language Reference*
- "S [partition by A1,..., Ak rows N range T1 slide T2]" in the *Oracle Complex Event Processing CQL Language Reference*

20.3.3.5 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 20.3.3.6, "Prefer Application Time"](#)
- "DURATION Clause" in the *Oracle Complex Event Processing CQL Language Reference*

20.3.3.6 Prefer Application Time

In Oracle CEP 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 CEP, 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 CEP high availability input adapter that intercepts incoming events and assigns a consistent time that spans primary and secondary instances.

Configuring High Availability

This chapter describes how to configure high availability for your Oracle CEP application depending on the quality of service you require, including:

- [Section 21.1, "Configuring High Availability Quality of Service"](#)
- [Section 21.2, "Configuring High Availability Adapters"](#)

For more information on high availability options, see [Chapter 20, "Understanding High Availability"](#).

21.1 Configuring High Availability Quality of Service

You configure Oracle CEP high availability quality of service in the EPN assembly file and component configuration files. For general information about these configuration files, see:

- [Section 1.1.4, "EPN Assembly File"](#)
- [Section 1.1.5, "Component Configuration Files"](#)

Note: After making any Oracle CEP high availability configuration changes, you must redeploy your Oracle CEP application. See [Section 24.5, "Deploying Oracle CEP Applications"](#).

This section describes:

- [Section 21.1.1, "How to Configure Simple Failover"](#)
- [Section 21.1.2, "How to Configure Simple Failover With Buffering"](#)
- [Section 21.1.3, "How to Configure Light-Weight Queue Trimming"](#)
- [Section 21.1.4, "How to Configure Precise Recovery With JMS"](#)

For more information on configuring an Oracle CEP high availability application for scalability, see [Section 20.1.4, "High Availability and Scalability"](#).

21.1.1 How to Configure Simple Failover

You configure simple failover using the Oracle CEP buffering output adapter with a sliding window size of zero (0).

This procedure starts with the example EPN that [Figure 21–1](#) shows and adds the required components to configure it for simple failover. [Example 21–1](#) shows the corresponding EPN assembly file and [Example 21–2](#) shows the corresponding component configuration file.

For more information about this Oracle CEP high availability quality of service, see [Section 20.2.1, "Simple Failover"](#).

Figure 21–1 Simple Failover EPN



Example 21–1 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 21–2 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 CEP Multi-Server Domain With Default Groups Using Oracle Coherence" in the *Oracle Complex Event Processing Administrator's Guide*.
 - "How to Create an Oracle CEP Multi-Server Domain With Custom Groups Using Oracle Coherence" in the *Oracle Complex Event Processing Administrator's Guide*.
2. Create an Oracle CEP application.
For more information, see [Section 4.2, "Creating Oracle CEP 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 4.7.2, "How to Add an OSGi Bundle to an Oracle CEP Project"](#).
4. Configure your Oracle CEP application EPN assembly file to add an Oracle CEP high availability buffering output adapter as [Example 21-3](#) 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 21-3 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>

```

- Optionally, configure the channel downstream from the input adapter (`helloworldInputChannel`) to configure an application timestamp based on an appropriate event property as [Example 21–4](#) 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 21–4 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>
...

```

- Configure the Oracle CEP high availability buffering output adapter. Set the instance property `windowLength` to zero (0) as [Example 21–5](#) shows.

Example 21–5 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 21.2.2.1, "Buffering Output Adapter EPN Assembly File Configuration"](#).

- Optionally, configure the component configuration file to include the Oracle CEP high availability buffering output adapter as [Example 21–6](#) shows.

Example 21–6 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 21.2.2.2, "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 24.5, "Deploying Oracle CEP Applications"](#).
Oracle CEP automatically selects one of the Oracle CEP servers as the primary.

21.1.2 How to Configure Simple Failover With Buffering

You configure simple failover using the Oracle CEP buffering output adapter with a sliding window size greater than zero (0).

This procedure starts with the example EPN that [Figure 21–2](#) shows and adds the required components to configure it for simple failover with buffering. [Example 21–7](#) shows the corresponding EPN assembly file and [Example 21–8](#) shows the corresponding component configuration file.

For more information about this Oracle CEP high availability quality of service, see [Section 20.2.2, "Simple Failover with Buffering"](#).

Figure 21–2 Simple Failover With Buffering EPN



Example 21–7 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: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 21–8 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 CEP Multi-Server Domain With Default Groups Using Oracle Coherence" in the *Oracle Complex Event Processing Administrator's Guide*.
- "How to Create an Oracle CEP Multi-Server Domain With Custom Groups Using Oracle Coherence" in the *Oracle Complex Event Processing Administrator's Guide*.

2. Create an Oracle CEP application.

For more information, see [Section 4.2, "Creating Oracle CEP 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 4.7.2, "How to Add an OSGi Bundle to an Oracle CEP Project"](#).

4. Configure your Oracle CEP application EPN assembly file to add an Oracle CEP high availability buffering output adapter as [Example 21–3](#) 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 21–9 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 21–10](#) 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 21–10 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 CEP high availability buffering output adapter.

Set the instance property `windowLength` to a value greater than zero (0) as [Example 21–11](#) shows.

Example 21–11 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 21.2.2.1, "Buffering Output Adapter EPN Assembly File Configuration"](#).

- Optionally, configure the component configuration file to include the Oracle CEP high availability buffering output adapter as [Example 21–12](#) shows.

Example 21–12 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 21.2.2.2, "Buffering Output Adapter Component Configuration File Configuration"](#)
- If your application is an Oracle CEP 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 20.3.2.5, "Choose an Adequate warm-up-window Time"](#)
 - [Section 21.2.2.2, "Buffering Output Adapter Component Configuration File Configuration"](#)
- Deploy your application to the deployment group you created in step 1.

For more information, see [Section 24.5, "Deploying Oracle CEP Applications"](#).

Oracle CEP automatically selects one of the Oracle CEP servers as the primary.

21.1.3 How to Configure Light-Weight Queue Trimming

You configure light-weight queue trimming using the Oracle CEP high availability input adapter and the broadcast output adapter.

This procedure starts with the example EPN that [Figure 21–3](#) shows and adds the required components to configure it for light-weight queue trimming. [Example 21–13](#) shows the corresponding EPN assembly file and [Example 21–14](#) shows the corresponding component configuration file.

For more information about this Oracle CEP high availability quality of service, see [Section 20.2.3, "Light-Weight Queue Trimming"](#).

Figure 21–3 Light-Weight Queue Trimming EPN



Example 21–13 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 21–14 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>

```

```

    </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 CEP Multi-Server Domain With Default Groups Using Oracle Coherence" in the *Oracle Complex Event Processing Administrator's Guide*.
- "How to Create an Oracle CEP Multi-Server Domain With Custom Groups Using Oracle Coherence" in the *Oracle Complex Event Processing Administrator's Guide*.

2. Create an Oracle CEP application.

For more information, see [Section 4.2, "Creating Oracle CEP 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 4.7.2, "How to Add an OSGi Bundle to an Oracle CEP Project"](#).

4. Configure your Oracle CEP application EPN assembly file to add an Oracle CEP high availability input adapter as [Example 21–15](#) 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 21–15 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 CEP application EPN assembly file to add an Oracle CEP high availability broadcast output adapter as [Example 21–16](#) 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 21–16 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 CEP high availability input adapter.

Consider the following example configurations:

- [Example 21–17, "High Availability Input Adapter: Default Configuration"](#)
- [Example 21–18, "High Availability Input Adapter: Tuple Events"](#)
- [Example 21–19, "High Availability Input Adapter: Key of One Event Property"](#)
- [Example 21–20, "High Availability Input Adapter: Key of Multiple Event Properties"](#)

For more information, see [Section 21.2.1.1, "High Availability Input Adapter EPN Assembly File Configuration"](#).

Example 21–17 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 21–18 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 21–19 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 21–20 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 21–21](#). The `hashCode` and `equals` methods are required. When you specify a `keyClass`, the `keyProperties` instance property is ignored: Oracle CEP 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 21–21 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 21–22](#) shows.

The `wlevs:expression` should be set to the `timeProperty` value.

Example 21–22 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 CEP high availability broadcast output adapter.

Consider the following example configurations:

- [Example 21–23, "Broadcast Output Adapter: Default Configuration"](#)
- [Example 21–24, "Broadcast Output Adapter: Key of One Event Property"](#)
- [Example 21–25, "Broadcast Output Adapter: Key of Multiple Event Properties"](#)

For more information, see [Section 21.2.3.1, "Broadcast Output Adapter EPN Assembly File Configuration"](#).

Example 21–23 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-buffering" >
  <wlevs:listener>
    <bean class="com.bea.wlevs.example.helloworld.HelloWorldBean"/>
  </wlevs:listener>
</wlevs:adapter>
...

```

Example 21–24 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-buffering" >
  <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="total-order" value="false"/>
</wlevs:adapter>
...

```

Example 21–25 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 21–26](#). The `hashCode` and `equals` methods are required. When you specify a `keyClass`, the `keyProperties` instance property is ignored: Oracle CEP assumes that the compound key is based on all the getter methods in the `keyClass`.

```
...
<wlevs:adapter id="myHaSlidingWindowAdapter" provider="ha-buffering" >
  <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="total-order" value="true"/>
</wlevs:adapter>
...
```

Example 21–26 *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 CEP high availability input adapter and buffering output adapter as [Example 21–27](#) shows.

Example 21–27 *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 21.2.1.2, "High Availability Input Adapter Component Configuration File Configuration"](#)
 - [Section 21.2.3.2, "Broadcast Output Adapter Component Configuration File Configuration"](#)
10. If your application is an Oracle CEP 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:

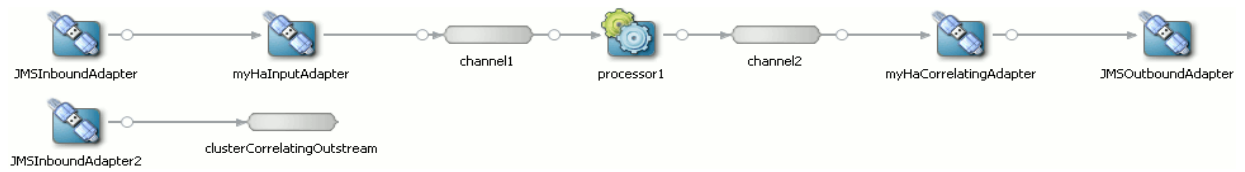
- [Section 20.3.2.5, "Choose an Adequate warm-up-window Time"](#)
 - [Section 21.2.3.2, "Broadcast Output Adapter Component Configuration File Configuration"](#)
11. Deploy your application to the deployment group you created in step 1.
- For more information, see [Section 24.5, "Deploying Oracle CEP Applications"](#).
- Oracle CEP automatically selects one of the Oracle CEP servers as the primary.

21.1.4 How to Configure Precise Recovery With JMS

You configure precise recovery with JMS using the Oracle CEP high availability input adapter and correlating output adapter.

This procedure describes how to create the example EPN that [Figure 21–4](#) shows. [Example 21–28](#) shows the corresponding EPN assembly file and [Example 21–29](#) shows the corresponding component configuration file.

For more information about this Oracle CEP high availability quality of service, see [Section 20.2.4, "Precise Recovery with JMS"](#).

Figure 21–4 Precise Recovery With JMS EPN**Example 21–28 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: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 21–29 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 CEP Multi-Server Domain With Default Groups Using Oracle Coherence" in the *Oracle Complex Event Processing Administrator's Guide*.
- "How to Create an Oracle CEP Multi-Server Domain With Custom Groups Using Oracle Coherence" in the *Oracle Complex Event Processing Administrator's Guide*.

2. Create an Oracle CEP application.

For more information, see [Section 4.2, "Creating Oracle CEP 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 4.7.2, "How to Add an OSGi Bundle to an Oracle CEP Project"](#).

4. Configure your Oracle CEP application EPN assembly file to add an Oracle CEP high availability input adapter as [Example 21–30](#) 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 21–30 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 CEP application EPN assembly file to add an Oracle CEP high availability correlating output adapter as [Example 21–31](#) 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 21–31 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: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 CEP high availability input adapter.

Consider the following example configurations:

- [Example 21–32, "High Availability Input Adapter: Default Configuration"](#)
- [Example 21–33, "High Availability Input Adapter: Tuple Events"](#)
- [Example 21–34, "High Availability Input Adapter: Key of One Event Property"](#)
- [Example 21–35, "High Availability Input Adapter: Key of Multiple Event Properties"](#)

For more information, see [Section 21.2.1.1, "High Availability Input Adapter EPN Assembly File Configuration"](#).

Example 21–32 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 21–33 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 21–34 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 21–35 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 21–36](#). The `hashCode` and `equals` methods are required. When you specify a `keyClass`, the `keyProperties` instance property is ignored: Oracle CEP 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 21–36 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;
    }
}

```

- 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 21–37](#) shows.

The `wlevs:expression` should be set to the `timeProperty` value.

Example 21–37 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>
...
```

- Configure the Oracle CEP high availability correlating output adapter `failOverDelay`.

[Example 21–38](#) shows a correlating output adapter configuration where the `failOverDelay` is 2000 milliseconds.

Example 21–38 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 21.2.4.1, "Correlating Output Adapter EPN Assembly File Configuration"](#).

- Create a second regular JMS input adapter.

[Example 21–39](#) shows a JMS adapter named `JMSInboundAdapter2`.

Example 21–39 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 21–40](#) shows the component configuration file for both the JMS input adapters. Note that both have exactly the same configuration, including the same provider.

Example 21–40 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>./Queue1</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>./Queue1</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 21–41](#) shows a correlated source named `clusterCorrelatingOutstream` whose source is `JMSInboundAdapter2`.

Example 21–41 *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 CEP high availability correlating output adapter with the `correlatedSource`.

[Example 21–38](#) shows a correlating output adapter configuration where the `correlatedSource` is `clusterCorrelatingOutstream`.

Example 21–42 *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 21.2.4.1, "Correlating Output Adapter EPN Assembly File Configuration"](#).

12. If your application is an Oracle CEP 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 20.3.2.5, "Choose an Adequate warm-up-window Time"](#)
- [Section 21.2.4.2, "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 21–43](#) shows:

Example 21–43 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>./Queue1</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>./Queue1</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 CEP high availability input adapter and correlating output adapter as [Example 21–27](#) shows.

Example 21–44 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 21.2.1.2, "High Availability Input Adapter Component Configuration File Configuration"](#)
 - [Section 21.2.4.2, "Correlating Output Adapter Component Configuration File Configuration"](#)
15. Optionally, add an `ActiveActiveGroupBean` to your EPN to improve scalability.
- For more information, see [Section 23.2, "Configuring Scalability With the `ActiveActiveGroupBean`"](#).
16. Deploy your application to the deployment group you created in step 1.
- For more information, see [Section 24.5, "Deploying Oracle CEP Applications"](#).
- Oracle CEP automatically selects one of the Oracle CEP servers as the primary.

21.2 Configuring High Availability Adapters

You configure Oracle CEP 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 1.1.4, "EPN Assembly File"](#)
- [Section 1.1.5, "Component Configuration Files"](#)

Note: After making any Oracle CEP high availability configuration changes, you must redeploy your Oracle CEP application. See [Section 24.5, "Deploying Oracle CEP Applications"](#).

This section describes the configurable options for each of the Oracle CEP high availability adapters, including:

- [Section 21.2.1, "How to Configure the High Availability Input Adapter"](#)
- [Section 21.2.2, "How to Configure the Buffering Output Adapter"](#)
- [Section 21.2.3, "How to Configure the Broadcast Output Adapter"](#)
- [Section 21.2.4, "How to Configure the Correlating Output Adapter"](#)

21.2.1 How to Configure the High Availability Input Adapter

The Oracle CEP high availability broadcast output adapter is implemented by `BroadcastInputAdapter`.

This section describes how to configure the Oracle CEP high availability input adapter, including:

- [Section 21.2.1.1, "High Availability Input Adapter EPN Assembly File Configuration"](#)
- [Section 21.2.1.2, "High Availability Input Adapter Component Configuration File Configuration"](#)

For more information, see [Section 20.1.3.1, "High Availability Input Adapter"](#).

21.2.1.1 High Availability Input Adapter EPN Assembly File Configuration

The root element for declaring an Oracle CEP high availability input adapter is `wlevs:adapter` with `provider` element set to `ha-inbound` as [Example 21-45](#) shows. You specify a `wlevs:listener` element for the Oracle CEP high availability input adapter in the actual input adapter as [Example 21-45](#) shows.

Example 21-45 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 21-1](#) describes the additional child elements of `wlevs:adapter` you can configure for an Oracle CEP high availability input adapter.

Table 21-1 Child Elements of `wlevs:adapter` for the High Availability Input Adapter

Child Element	Description
<code>wlevs:instance-property</code>	Specify one or more <code>instance-property</code> element name and <code>value</code> attributes as Table 21-2 describes.

[Table 21-2](#) lists the instance properties that the Oracle CEP high availability input adapter supports.

Table 21-2 High Availability Input Adapter Instance Properties

Name	Value
<code>timeProperty</code>	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 <code>wlevs:application-timestamped</code> element of the downstream EPN component to which the high availability input adapter is connected as Example 21-45 shows.

Table 21–2 (Cont.) High Availability Input Adapter Instance Properties

Name	Value
keyProperties	Specify a space delimited list of one or more event properties that the Oracle CEP high availability input adapter uses to identify event instances. If you specify more than one event property, you must specify a keyClass . 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 <code>keyClass</code> are assumed to be <code>keyProperties</code> , unless the <code>keyProperties</code> setting is used.
eventType	Specify the type name of the events that the Oracle CEP 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 Example 21–45 shows. For tuple events, this property is mandatory. For all other Java class-based event types, this property is optional. For more information, see Section 1.1.2, "Oracle CEP Event Types" .

21.2.1.2 High Availability Input Adapter Component Configuration File Configuration

The root element for configuring an Oracle CEP 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 21–50](#) shows.

Example 21–46 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 21–3](#) describes the additional child elements of `ha-inbound-adapter` you can configure for an Oracle CEP high availability input adapter.

Table 21–3 Child Elements of `ha-inbound-adapter` for the High Availability Input Adapter

Child Element	Description
heartbeat	Specify the length of time that the Oracle CEP high availability input adapter can be idle before it generates a heartbeat event to advance time as an integer number of <code>units</code> . Valid values for attribute <code>units</code> : <ul style="list-style-type: none"> ▪ <code>nanos</code>: wait the specified number of nanoseconds. ▪ <code>millis</code>: wait the specified number of milliseconds. ▪ <code>secs</code>: wait the specified number of seconds. Default: Heartbeats are not sent.

Table 21–3 (Cont.) Child Elements of ha-inbound-adapter for the High Availability Input

Child Element	Description
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 Section 27.2.3, "High Availability Input Adapter Configuration") Default: 1 (disable batching).

21.2.2 How to Configure the Buffering Output Adapter

The Oracle CEP high availability buffering output adapter is implemented by `SlidingWindowQueueTrimmingAdapter`.

This section describes how to configure the Oracle CEP high availability buffering output adapter, including:

- [Section 21.2.2.1, "Buffering Output Adapter EPN Assembly File Configuration"](#)
- [Section 21.2.2.2, "Buffering Output Adapter Component Configuration File Configuration"](#)

For more information, see [Section 20.1.3.2, "Buffering Output Adapter"](#).

21.2.2.1 Buffering Output Adapter EPN Assembly File Configuration

The root element for declaring an Oracle CEP high availability buffering output adapter is `wlevs:adapter` with `provider` element set to `ha-buffering` as [Example 21–47](#) shows.

Example 21–47 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 21–4](#) describes the additional child elements of `wlevs:adapter` you can configure for an Oracle CEP high availability buffering output adapter.

Table 21–4 Child Elements of wlevs:adapter for the Buffering Output Adapter

Child Element	Description
wlevs:listener	Specify the regular output adapter downstream from this Oracle CEP high availability buffering output adapter.
wlevs:instance-property	Specify one or more <code>instance-property</code> element name and value attributes as Table 21–5 describes.

[Table 21–5](#) lists the instance properties that the Oracle CEP high availability broadcast output adapter supports.

Table 21–5 Buffering Output Adapter Instance Properties

Name	Value
windowLength	Specify the size of the sliding window as an integer number of milliseconds. Default: 15000.

21.2.2.2 Buffering Output Adapter Component Configuration File Configuration

The root element for configuring an Oracle CEP 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 21–48](#) shows.

Example 21–48 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 21–6](#) describes the additional child elements of `ha-buffering-adapter` you can configure for an Oracle CEP high availability buffering output adapter.

Table 21–6 Child Elements of `ha-buffering-adapter` for the Buffering Output Adapter

Child Element	Description
<code>window-length</code>	Specify the size of the sliding window as an integer number of milliseconds. Default: 15000.
<code>warm-up-window</code>	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"> ▪ <code>seconds</code>: wait the specified number of seconds. ▪ <code>minutes</code>: wait the specified number of minutes. Default: <code>units</code> is events. For more information, see Section 20.3.2.5, "Choose an Adequate warm-up-window Time" .

21.2.3 How to Configure the Broadcast Output Adapter

The Oracle CEP high availability broadcast output adapter is implemented by class `GroupBroadcastQueueTrimmingAdapter`.

This section describes how to configure the Oracle CEP high availability broadcast output adapter, including:

- [Section 21.2.3.1, "Broadcast Output Adapter EPN Assembly File Configuration"](#)
- [Section 21.2.3.2, "Broadcast Output Adapter Component Configuration File Configuration"](#)

For more information, see [Section 20.1.3.3, "Broadcast Output Adapter"](#).

21.2.3.1 Broadcast Output Adapter EPN Assembly File Configuration

The root element for declaring an Oracle CEP high availability broadcast output adapter is `wlevs:adapter` with `provider` element set to `ha-broadcast` as [Example 21–49](#) shows.

Example 21–49 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 21-7 describes the additional child elements of `wlevs:adapter` you can configure for an Oracle CEP high availability broadcast output adapter.

Table 21-7 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 CEP high availability broadcast output adapter.
<code>wlevs:instance-property</code>	Specify one or more <code>instance-property</code> element name and <code>value</code> attributes as Table 21-8 describes.

Table 21-8 lists the instance properties that the Oracle CEP high availability broadcast output adapter supports.

Table 21-8 Broadcast Output Adapter Instance Properties

Name	Value
<code>keyProperties</code>	Specify a space delimited list of one or more event properties that the Oracle CEP 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>total-order</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"> ▪ <code>true</code>: the key is constantly increasing. ▪ <code>false</code>: the key is not constantly increasing. Default: <code>false</code> .
<code>total-order</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"> ▪ <code>true</code>: event keys are unique. ▪ <code>false</code>: event keys are not unique. Default: <code>true</code> .

21.2.3.2 Broadcast Output Adapter Component Configuration File Configuration

The root element for configuring an Oracle CEP 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 21-50 shows.

Example 21-50 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 21–9 describes the additional child elements of `ha-broadcast-adapter` you can configure for an Oracle CEP high availability broadcast output adapter.

Table 21–9 Child Elements of `ha-broadcast-adapter` for the Broadcast Output Adapter

Child Element	Description
<code>trimming-interval</code>	<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 Section 27.2.4, "Broadcast Output Adapter Configuration").</p> <p>Valid values for attribute units:</p> <ul style="list-style-type: none"> events: broadcast trimming messages after the specified number of milliseconds. millis: broadcast trimming messages after the specified number of events are processed. <p>Default: units is events.</p>
<code>warm-up-window</code>	<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"> seconds: wait the specified number of seconds. minutes: wait the specified number of minutes. <p>Default: units is events.</p> <p>For more information, see Section 20.3.2.5, "Choose an Adequate warm-up-window Time".</p>

21.2.4 How to Configure the Correlating Output Adapter

The Oracle CEP high availability correlating output adapter is implemented by class `CorrelatedQueueTrimmingAdapter`.

This section describes how to configure the Oracle CEP high availability correlating output adapter, including:

- Section 21.2.4.1, "Correlating Output Adapter EPN Assembly File Configuration"
- Section 21.2.4.2, "Correlating Output Adapter Component Configuration File Configuration"

For more information, see Section 20.1.3.4, "Correlating Output Adapter".

21.2.4.1 Correlating Output Adapter EPN Assembly File Configuration

The root element for declaring an Oracle CEP high availability correlating output adapter is `wlevs:adapter` with `provider` element set to `ha-correlating` as Example 21–51 shows.

Example 21–51 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 21–10](#) describes the additional child elements of `wlevs:adapter` you can configure for an Oracle CEP high availability correlating output adapter.

Table 21–10 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 CEP 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 21–11 describes.

[Table 21–11](#) lists the instance properties that the Oracle CEP high availability correlating output adapter supports.

Table 21–11 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.

21.2.4.2 Correlating Output Adapter Component Configuration File Configuration

The root element for configuring an Oracle CEP 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 21–52](#) shows.

Example 21–52 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 21–12](#) describes the additional child elements of `ha-broadcast-adapter` you can configure for an Oracle CEP high availability correlating output adapter.

Table 21–12 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.
<code>warm-up-window</code>	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 <code>units</code> : <ul style="list-style-type: none"> ▪ <code>seconds</code>: wait the specified number of seconds. ▪ <code>minutes</code>: wait the specified number of minutes. Default: <code>units</code> is <code>events</code> . For more information, see Section 20.3.2.5, "Choose an Adequate warm-up-window Time" .

Part VI

Developing Applications for Scalability

Part VI contains the following chapters:

- [Chapter 22, "Understanding Scalability"](#)
- [Chapter 23, "Configuring Scalability"](#)

Understanding Scalability

This chapter describes Oracle CEP components and design patterns that you can use to allow your Oracle CEP application to scale with an increasing event load, including:

- [Section 22.1, "Scalability Options"](#)
- [Section 22.2, "Scalability Components"](#)

For more information on how to implement a particular scalability option, see [Chapter 23, "Configuring Scalability"](#).

22.1 Scalability Options

Oracle CEP provides options that you can use to allow your Oracle CEP 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 CEP provides a variety of scalability components you can use as [Section 22.2, "Scalability Components"](#) describes.

22.1.1 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 CEP 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 20.1, "High Availability Architecture"](#)
- [Section 20.3, "Designing an Oracle CEP Application for High Availability"](#)

22.2 Scalability Components

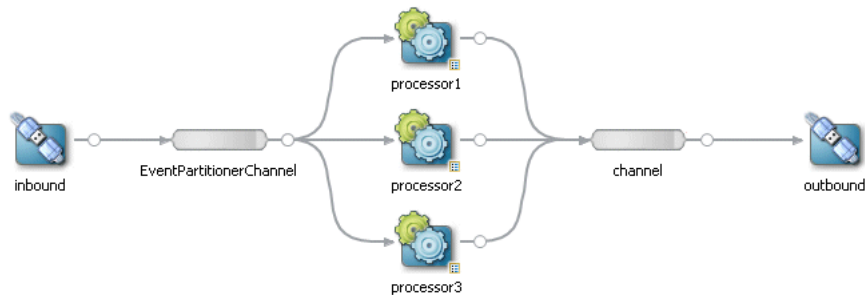
Oracle CEP provides the following components that you can use to improve the scalability of your Oracle CEP applications:

- [Section 22.2.1, "EventPartitioner"](#)
- [Section 22.2.2, "ActiveActiveGroupBean"](#)

22.2.1 EventPartitioner

A `com.bea.wlevs.channel.EventPartitioner` provides a mechanism for partitioning events on a channel across its output event sinks as [Figure 22–1](#) shows.

Figure 22–1 Event Partitioner EPN



This section describes:

- [Section 22.2.1.1, "EventPartitioner Implementation"](#)
- [Section 22.2.1.2, "EventPartitioner Load Balancing"](#)
- [Section 22.2.1.3, "EventPartitioner Initialization"](#)
- [Section 22.2.1.4, "EventPartitioner Threading"](#)
- [Section 22.2.1.5, "EventPartitioner Restrictions"](#)

For more information, see [Section 23.1, "Configuring Scalability With a Channel EventPartitioner"](#).

22.2.1.1 EventPartitioner Implementation

Oracle CEP 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 23.1.1, "How to Configure Scalability With the Default Channel EventPartitioner"](#)
- [Section 23.1.2, "How to Configure Scalability With a Custom Channel EventPartitioner"](#)

22.2.1.2 EventPartitioner Load Balancing

When using an event partitioner channel, if you want to perform load balancing, then each listener must be identical.

Otherwise, listeners need not be identical.

22.2.1.3 EventPartitioner Initialization

By default, the Oracle CEP 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.

22.2.1.4 EventPartitioner Threading

[Table 22–1](#) lists the threading options you can use with an event partitioner channel.

Table 22–1 *Event Partitioner Channel Threading Options*

Threads Allocated In	Description	When to Use
Channel	<ul style="list-style-type: none"> ■ Channel <code>max-threads</code> set to the number of listeners. 	<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"> ■ Channel <code>max-threads</code> set to 0. ■ Implement a multi-threaded adapter with at least on thread per partition listener. 	<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.

22.2.1.5 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 9.1.6, "Batch Processing Channels"](#).

22.2.2 ActiveActiveGroupBean

Using the `com.oracle.cep.cluster.hagroups.ActiveActiveGroupBean`, you can partition an incoming JMS stream in Oracle CEP applications by utilizing the notification groups that the `ActiveActiveGroupBean` creates.

You add an `ActiveActiveGroupBean` to your EPN assembly file as [Example 22–1](#) shows.

Example 22–1 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 CEP 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 23.2.3, "How to Configure the ActiveActiveGroupBean Group Pattern Match"](#) describes.

This section describes:

- [Section 22.2.2.1, "Scalability in an Oracle CEP Application Using the ActiveActiveGroupBean Without High Availability"](#)
- [Section 22.2.2.2, "Scalability in an Oracle CEP Application Using the ActiveActiveGroupBean With High Availability"](#)

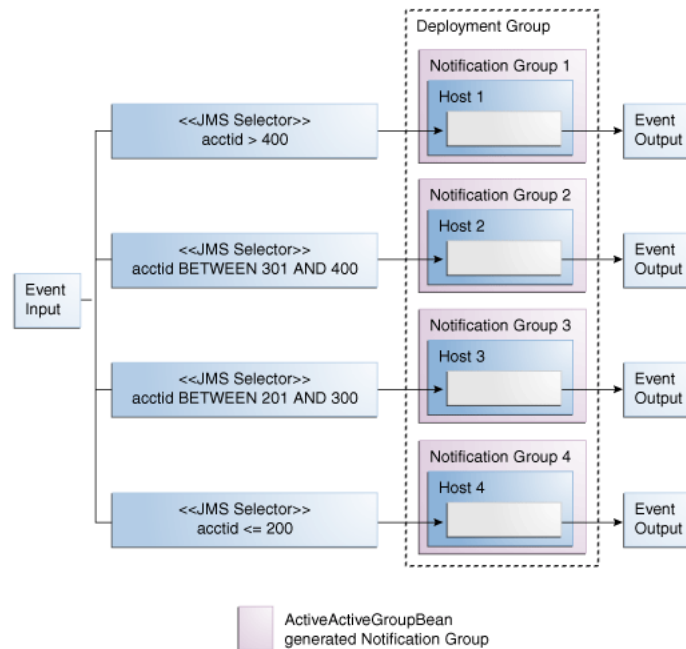
For more information, see:

- [Section 20.1.2, "Deployment Group and Notification Group"](#)
- [Section 23.2, "Configuring Scalability With the ActiveActiveGroupBean"](#)

22.2.2.1 Scalability in an Oracle CEP Application Using the ActiveActiveGroupBean Without High Availability

You can use the `ActiveActiveGroupBean` to partition an incoming JMS event stream by selector in an Oracle CEP application that is not configured for high availability.

Consider the multi-server domain that [Figure 22–2](#) shows.

Figure 22–2 Oracle CEP ActiveActiveGroupBean Without High Availability

In this scalability scenario, you define a cluster group in the Oracle CEP 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 CEP 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 CEP application contains the same `jms-adapter` configuration as [Example 22–2](#) shows.

Example 22–2 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 CEP application instance on each Oracle CEP server finds its `ActiveActiveGroupBean_` cluster group and creates a notification group based on it. The Oracle CEP 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 22.2.1, "EventPartitioner"](#) describes.

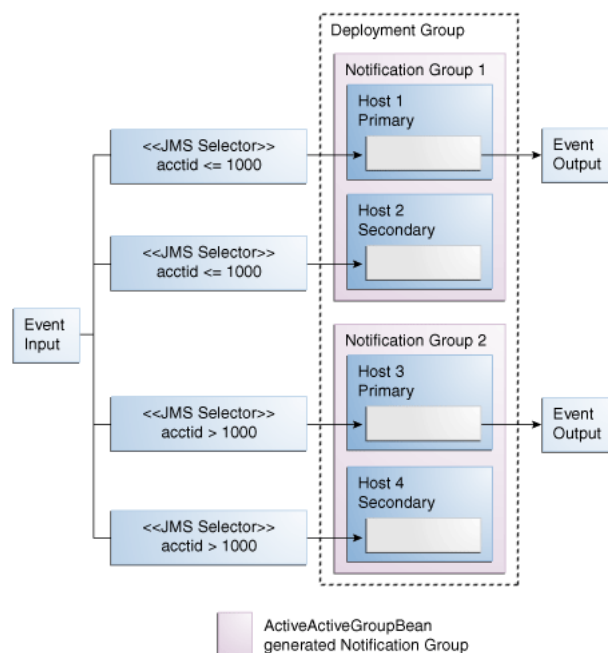
For more information, see [Section 23.2.1, "How to Configure Scalability in a JMS Application Without Oracle CEP High Availability"](#).

22.2.2.2 Scalability in an Oracle CEP 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 CEP servers to function as a single, high availability unit.

Consider the multi-server domain with an Oracle CEP high availability application deployed to it that [Figure 22-3](#) shows.

Figure 22-3 Oracle CEP 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 CEP application instance on each Oracle CEP 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 CEP 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 CEP 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 CEP 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 CEP 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 22.2.1, "EventPartitioner"](#) describes.

For more information, see [Section 23.2.2, "How to Configure Scalability in a JMS Application With Oracle CEP High Availability"](#).

Configuring Scalability

This chapter describes how to configure scalability for your Oracle CEP application depending on the quality of service you have selected, including:

- [Section 23.1, "Configuring Scalability With a Channel EventPartitioner"](#)
- [Section 23.2, "Configuring Scalability With the ActiveActiveGroupBean"](#)

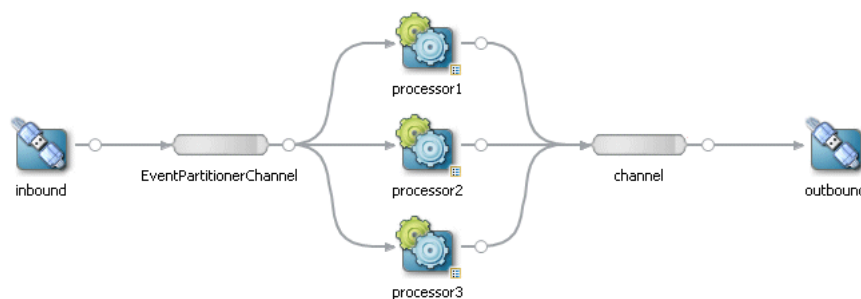
For more information on scalability options, see [Chapter 22, "Understanding Scalability"](#).

23.1 Configuring Scalability With a Channel EventPartitioner

This section describes how to configure a channel with an Oracle CEP event partitioner as [Figure 23–1](#) shows, including:

- [Section 23.1.1, "How to Configure Scalability With the Default Channel EventPartitioner"](#)
- [Section 23.1.2, "How to Configure Scalability With a Custom Channel EventPartitioner"](#)

Figure 23–1 *EventPartitioner EPN*



In this example, assume that the inbound adapter is sending events of type `PriceEvent`, defined as [Example 23–1](#) shows:

Example 23–1 *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>
  
```

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

For more information, see [Section 22.2.1, "EventPartitioner"](#).

23.1.1 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 23.1.2, "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 23–1](#), the channel is `EventPartitionerChannel`.

For more information, see [Chapter 9, "Configuring Channels"](#).

2. Connect the channel to an upstream adapter.

In [Figure 23–1](#), the upstream adapter is `inbound`.

For more information, see [Section 6.4.2, "Connecting Nodes"](#).

3. Connect the channel to two or more listeners.

In [Figure 23–1](#), 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 22.2.1.2, "EventPartitioner Load Balancing"](#)
- [Section 6.4.2, "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 22.2.1.1, "EventPartitioner Implementation"](#).

5. Decide how you want Oracle CEP to allocate threads as [Section 22.2.1.4, "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 CEP 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 F.44, "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>
    ...
```

6. Assemble and deploy your application.

For more information, see [Chapter 24, "Assembling and Deploying Oracle CEP Applications"](#).

At runtime, the channel uses the default event property-based `EventPartitioner` to determine how to dispatch each incoming event to its listeners.

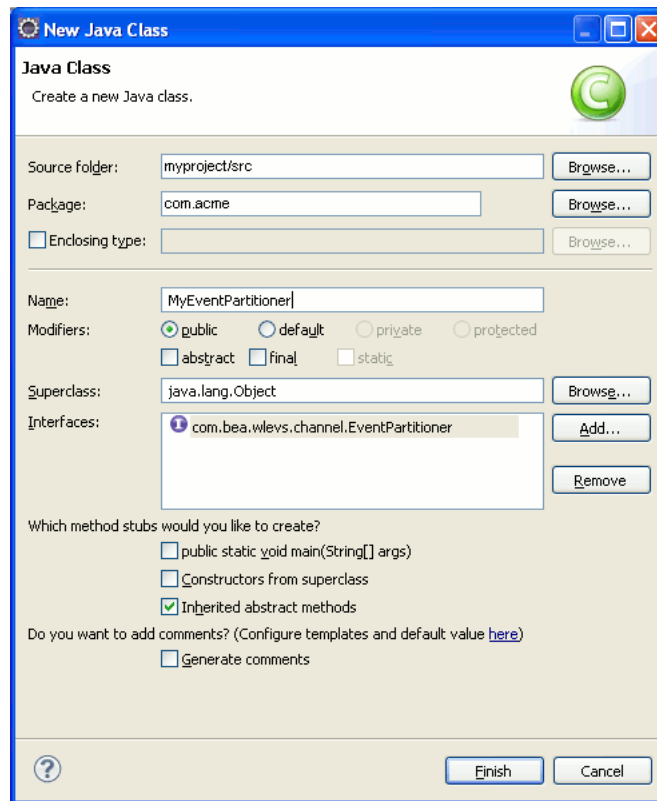
23.1.2 How to Configure Scalability With a Custom Channel EventPartitioner

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 23.1.1, "How to Configure Scalability With the Default Channel EventPartitioner"](#) describes.

To configure scalability with a custom channel EventPartitioner:

1. Using the Oracle CEP IDE for Eclipse, open your Oracle CEP project.
For more information, see [Section 4.2, "Creating Oracle CEP Projects"](#).
2. Edit your `MANIFEST.MF` to import package `com.bea.wlevs.channel`.
For more information, see [Section 4.7.5, "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 23–2](#) shows.

Figure 23–2 *New Java Class Dialog*

4. Configure the New Java Class dialog as [Table 23–1](#) describes.

Table 23–1 *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 Matching Items list, and then click OK .

5. Click **Finish**.

A new `EventPartitioner` class is created as [Example 23–2](#) shows.

Example 23–2 *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 23–3](#) shows.

Example 23–3 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 CEP 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 23–1](#), the channel is `EventPartitionerChannel`.

For more information, see [Chapter 9, "Configuring Channels"](#).

8. Connect the channel to an upstream adapter.

In [Figure 23–1](#), the upstream adapter is `inbound`.

For more information, see [Section 6.4.2, "Connecting Nodes"](#).

9. Connect the channel to two or more listeners.

In [Figure 23-1](#), 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 22.2.1.2, "EventPartitioner Load Balancing"](#)
- [Section 6.4.2, "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 CEP 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 CEP to allocate threads as [Section 22.2.1.4, "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 CEP 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 F.44, "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 24, "Assembling and Deploying Oracle CEP Applications"](#).

At runtime, the channel uses your `EventPartitioner` to determine how to dispatch each incoming event to its listeners.

23.2 Configuring Scalability With the ActiveActiveGroupBean

This section describes how to configure your Oracle CEP application to use the `ActiveActiveGroupBean` to partition an incoming JMS event stream by selector, including:

- [Section 23.2.1, "How to Configure Scalability in a JMS Application Without Oracle CEP High Availability"](#)
- [Section 23.2.2, "How to Configure Scalability in a JMS Application With Oracle CEP High Availability"](#)

- [Section 23.2.3, "How to Configure the ActiveActiveGroupBean Group Pattern Match"](#)

For more information, see [Section 22.2.2, "ActiveActiveGroupBean"](#).

23.2.1 How to Configure Scalability in a JMS Application Without Oracle CEP 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 CEP high availability.

For information on how to use the `ActiveActiveGroupBean` in an Oracle CEP high availability application, see [Section 23.2.2, "How to Configure Scalability in a JMS Application With Oracle CEP High Availability"](#).

For more information, see [Section 22.2.2.1, "Scalability in an Oracle CEP Application Using the ActiveActiveGroupBean Without High Availability"](#).

To configure scalability in a JMS application without Oracle CEP high availability:

1. Create a multi-server domain.

For more information, see "Introduction to Multi-Server Domains" in the *Oracle Complex Event Processing Administrator's Guide*.

In this example, the deployment group is named `MyDeploymentGroup`.

2. Configure the Oracle CEP server configuration file on each Oracle CEP server to add the appropriate `ActiveActiveGroupBean` notification group to the `groups` child element of the `cluster` element.

The Oracle CEP server configuration file, `config.xml`, is located in the `DOMAIN_DIR/servername/config` directory, where `DOMAIN_DIR` refers to the main domain directory and `servername` refers to a particular server instance.

For example, [Table 23–3](#) shows `cluster` elements for Oracle CEP 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 23.2.3, "How to Configure the ActiveActiveGroupBean Group Pattern Match"](#) describes.

Table 23–2 Oracle CEP Server Configuration File groups Element Configuration

Partition	cluster Element
ocep-server-1	<pre><cluster> <server-name>ocep-server-1</server-name> ... <enabled>coherence</enabled> ... <groups>MyDeploymentGroup, ActiveActiveGroupBean_group1</groups> </cluster></pre>
ocep-server-2	<pre><cluster> <server-name>ocep-server-2</server-name> ... <enabled>coherence</enabled> ... <groups>MyDeploymentGroup, ActiveActiveGroupBean_group2</groups> </cluster></pre>

Table 23–2 (Cont.) Oracle CEP Server Configuration File groups Element Configuration

Partition	cluster Element
ocep-server-3	<pre><cluster> <server-name>ocep-server-3</server-name> ... <enabled>coherence</enabled> ... <groups>MyDeploymentGroup, ActiveActiveGroupBean_group3</groups> </cluster></pre>
ocep-server-4	<pre><cluster> <server-name>ocep-server-4</server-name> ... <enabled>coherence</enabled> ... <groups>MyDeploymentGroup, ActiveActiveGroupBean_group4</groups> </cluster></pre>

3. Create an Oracle CEP application.
4. Configure the EPN assembly file to add an ActiveActiveGroupBean element as [Example 23–8](#) shows.

Example 23–4 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 23–10](#) shows.

Example 23–5 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>
```



```
</jms-adapter>
```

In this configuration, when the application is deployed to an Oracle CEP 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 7, "Configuring JMS Adapters"](#).

6. Deploy your application to the deployment group of your multi-server domain.

For more information, see [Chapter 24, "Assembling and Deploying Oracle CEP Applications"](#).

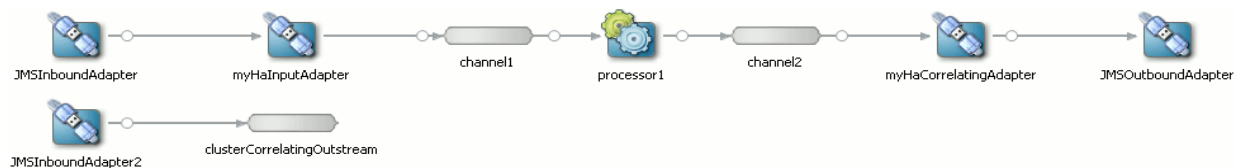
At runtime, each Oracle CEP server configures its instance of the application with the `message-selector` that corresponds to its `ActiveActiveGroupBean` notification group. This partitions the JMS topic so that each instance of the application processes a subset of the total number of messages in parallel.

23.2.2 How to Configure Scalability in a JMS Application With Oracle CEP High Availability

You can use the `ActiveActiveGroupBean` to partition an incoming JMS event stream in a multi-server domain with Oracle CEP high availability.

This procedure uses the example application from [Section 21.1.4, "How to Configure Precise Recovery With JMS"](#), including the example EPN that [Figure 23–3](#) shows, the corresponding EPN assembly file that [Example 23–6](#) shows, and the corresponding component configuration file that [Example 23–7](#) shows.

Figure 23–3 *Precise Recovery With JMS EPN*



Example 23–6 *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 23–7 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>

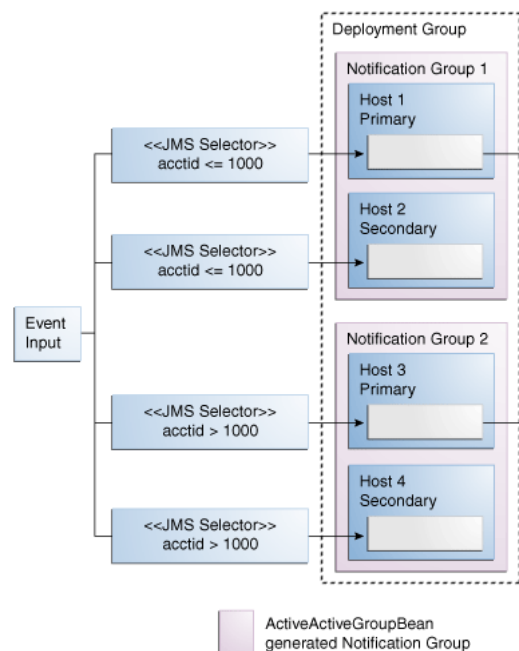
```

```

        <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 CEP high availability configuration that [Figure 23–4](#) shows.

Figure 23–4 Oracle CEP ActiveActiveGroupBean With High Availability



For more information, see [Section 22.2.2.2, "Scalability in an Oracle CEP Application Using the ActiveActiveGroupBean With High Availability"](#).

To configure scalability in a JMS application with Oracle CEP high availability:

1. Create a multi-server domain.

For more information, see "Introduction to Multi-Server Domains" in the *Oracle Complex Event Processing Administrator's Guide*.

In this example, the deployment group is named `MyDeploymentGroup`.

2. Configure the Oracle CEP server configuration file on each Oracle CEP server to add the appropriate ActiveActiveGroupBean notification group to the groups child element of the cluster element.

The Oracle CEP server configuration file, `config.xml`, is located in the `DOMAIN_DIR/servername/config` directory, where `DOMAIN_DIR` refers to the main domain directory and `servername` refers to a particular server instance.

For example, [Table 23-3](#) shows cluster elements for Oracle CEP 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 23.2.3, "How to Configure the ActiveActiveGroupBean Group Pattern Match"](#) describes.

Table 23-3 Oracle CEP Server Configuration File groups Element Configuration

Partition	cluster Element
ocep-server-1	<pre><cluster> <server-name>ocep-server-1</server-name> ... <enabled>coherence</enabled> ... <groups>MyDeploymentGroup, ActiveActiveGroupBean_group1</groups> </cluster></pre>
ocep-server-2	<pre><cluster> <server-name>ocep-server-2</server-name> ... <enabled>coherence</enabled> ... <groups>MyDeploymentGroup, ActiveActiveGroupBean_group1</groups> </cluster></pre>
ocep-server-3	<pre><cluster> <server-name>ocep-server-3</server-name> ... <enabled>coherence</enabled> ... <groups>MyDeploymentGroup, ActiveActiveGroupBean_group2</groups> </cluster></pre>
ocep-server-4	<pre><cluster> <server-name>ocep-server-4</server-name> ... <enabled>coherence</enabled> ... <groups>MyDeploymentGroup, ActiveActiveGroupBean_group2</groups> </cluster></pre>

3. Create an Oracle CEP high availability application.
For more information, see [Chapter 21, "Configuring High Availability"](#).
4. Configure the EPN assembly file to add an ActiveActiveGroupBean element as [Example 23-8](#) shows.

Example 23–8 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 23–9](#) shows:
 - Each in-bound JMS adapter must listen to a different topic.
 - Set `session-transacted` to `true`.

Example 23–9 `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 7, "Configuring JMS Adapters"](#).

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 23–10](#) shows.

Example 23–10 `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>
</jms-adapter>
```

In this configuration, when the application is deployed to an Oracle CEP 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 CEP 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 23–11](#) 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 23–11 `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 7, "Configuring JMS Adapters"](#).

8. Deploy your application to the deployment group of your multi-server domain.

For more information, see [Chapter 24, "Assembling and Deploying Oracle CEP Applications"](#).

At runtime, each Oracle CEP 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 CEP server in an `ActiveActiveGroupBean` group goes down, the Oracle CEP server performs an Oracle CEP high availability failover to the standby Oracle CEP server in that `ActiveActiveGroupBean` group.

23.2.3 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 CEP 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 23–12](#) shows.

Example 23–12 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.

Part VII

Assembly, Deployment, and Testing

Part VII contains the following chapters:

- [Chapter 24, "Assembling and Deploying Oracle CEP Applications"](#)
- [Chapter 25, "Testing Applications With the Load Generator and csvgen Adapter"](#)
- [Chapter 26, "Testing Applications With the Event Inspector"](#)
- [Chapter 27, "Performance Tuning"](#)

Assembling and Deploying Oracle CEP Applications

This section contains information on the following subjects:

- [Section 24.1, "Overview of Application Assembly and Deployment"](#)
- [Section 24.2, "Assembling an Oracle CEP Application"](#)
- [Section 24.3, "Managing Application Libraries"](#)
- [Section 24.4, "Managing Log Message Catalogs"](#)
- [Section 24.5, "Deploying Oracle CEP Applications"](#)

24.1 Overview of Application Assembly and Deployment

The term *application assembly* refers to the process of packaging the components of an application, such as the Java files and XML configuration files, into an OSGi bundle that can be deployed to Oracle CEP. The term *application deployment* refers to the process of making an application available for processing client requests in an Oracle CEP domain.

This section describes:

- [Section 24.1.1, "Applications"](#)
- [Section 24.1.3, "Application Libraries"](#)
- [Section 24.1.2, "Application Dependencies"](#)
- [Section 24.1.4, "Deployment and Deployment Order"](#)
- [Section 24.1.5, "Configuration History Management"](#)

Note: Oracle CEP applications are built on top of the Spring Framework and OSGi Service Platform and make extensive use of their technologies and services. See [Appendix A, "Additional Information about Spring and OSGi,"](#) for links to reference and conceptual information about Spring and OSGi.

24.1.1 Applications

In the context of Oracle CEP assembly and deployment, an application is defined as an OSGi bundle (see <http://www2.osgi.org/javadoc/r4/org/osgi/framework/Bundle.html>) JAR file that contains the following artifacts:

- The compiled Java class files that implement some of the components of the application, such as the adapters, adapter factory, and POJO that contains the business logic.
- One or more Oracle CEP configuration XML files that configure the components of the application. The only type of component that is required to have a configuration file is the complex event processor; all other components (adapters and streams) do not require configuration files if the default configuration of the component is adequate. You can combine all configuration files into a single file, or separate the configuration for individual components in their own files.

The configuration files must be located in the `META-INF/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.

24.1.2 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 24.1.2.1, "Private Application Dependencies"](#)
- [Section 24.1.2.2, "Shared Application Dependencies"](#)
- [Section 24.1.2.3, "Native Code Dependencies"](#)

For more information, see:

- [Section 24.1.4, "Deployment and Deployment Order"](#)
- [Section 24.2, "Assembling an Oracle CEP Application"](#)

24.1.2.1 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 4.7.1, "How to Add a Standard JAR File to an Oracle CEP Project"](#)
- [Section 4.7.2, "How to Add an OSGi Bundle to an Oracle CEP Project"](#)
- [Section 4.7.3, "How to Add a Property File to an Oracle CEP Project"](#)

24.1.2.2 Shared Application Dependencies

Some dependencies are satisfied by a component deployed to the Oracle CEP 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 24.1.2.3, "Native Code Dependencies"](#)
- [Section 24.1.3, "Application Libraries"](#)
- [Section 4.7.4, "How to Export a Package"](#)
- [Section 4.7.5, "How to Import a Package"](#)

24.1.2.3 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 CEP 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 24.1.2.2, "Shared Application Dependencies"](#)
- "Configuring the Oracle CEP Server Boot Classpath" in the *Oracle Complex Event Processing Administrator's Guide*

24.1.3 Application Libraries

The Oracle CEP 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 CEP 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 CEP 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 CEP server directories:

- [Section 24.1.3.1, "Library Directory"](#)
- [Section 24.1.3.2, "Library Extensions Directory"](#)
- [Section 24.1.3.3, "Creating Application Libraries"](#)

For more information, see:

- [Section 24.3, "Managing Application Libraries"](#)
- [Section 24.1.2, "Application Dependencies"](#)

- [Section 24.1.4, "Deployment and Deployment Order"](#)
- [Section 1.1.1.9, "Foreign Stages"](#)
- [Appendix A, "Additional Information about Spring and OSGi"](#)

24.1.3.1 Library Directory

By default, the Oracle CEP server library directory is:

```
DOMAIN_DIR/servername/modules
```

Where:

- *DOMAIN_DIR*: is the 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 CEP applications.

If your library is a driver (such as a JDBC driver), you must put it in the library extensions directory as [Section 24.1.3.2, "Library Extensions Directory"](#) describes.

To configure the root of the application library directory path, see [Section 24.3.1, "How to Define the Application Library Directory Using Oracle CEP IDE for Eclipse"](#).

24.1.3.2 Library Extensions Directory

By default, the Oracle CEP server library extensions directory is:

```
DOMAIN_DIR/servername/modules/ext
```

Where:

- *DOMAIN_DIR*: is the 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 CEP 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 Complex Event Processing Administrator's Guide*.

If your library is not a driver, you may put it in the library directory as [Section 24.1.3.1, "Library Directory"](#) describes.

To configure the root of the application library extensions directory path, see [Section 24.3.1, "How to Define the Application Library Directory Using Oracle CEP IDE for Eclipse"](#).

24.1.3.3 Creating Application Libraries

Oracle CEP 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 CEP IDE for Eclipse.

For more information, see:

- [Section 24.3.2, "How to Create an Application Library Using `bundler.sh`"](#)
- [Section 24.3.3, "How to Create an Application Library Using Oracle CEP IDE for Eclipse"](#)
- "How to Access a Database Driver Using an Application Library Built With `bundler.sh`" in the *Oracle Complex Event Processing Administrator's Guide*

24.1.4 Deployment and Deployment Order

After you have assembled the application, you deploy it by making it known to the Oracle CEP domain using the deployment tool appropriate for your needs. For detailed instructions, see [Section 24.5, "Deploying Oracle CEP Applications."](#)

The Oracle CEP server deploys components in the following order at Oracle CEP 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 CEP applications.

The Oracle CEP 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 CEP 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 CEP, 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 24.1.3, "Application Libraries"](#).

24.1.5 Configuration History Management

When you deploy an application to the Oracle CEP server, the Oracle CEP server creates a configuration history for the application. Any configuration changes you make to rules or Oracle CEP high availability adapter configuration are recorded in this history. You can view and roll-back (undo) these changes using the Oracle CEP Visualizer or `wlevs.Admin` tool.

For more information, see:

- "Configuration History Management" in the *Oracle Complex Event Processing Visualizer User's Guide*
- "Commands for Managing Configuration History" in the *Oracle Complex Event Processing Administrator's Guide*

24.2 Assembling an Oracle CEP Application

Assembling an Oracle CEP application refers to bundling the artifacts that make up the application into an OSGi bundle JAR file as

<http://www2.osgi.org/javadoc/r4/org/osgi/framework/Bundle.html> describes. These artifacts include:

- compiled Java classes
- Oracle CEP component configuration files that configure application components (such as the processors or adapters)
- EPN assembly file
- `MANIFEST.MF` file

See [Appendix A, "Additional Information about Spring and OSGi,"](#) for links to reference and conceptual information about Spring and OSGi.

This section describes:

- [Section 24.2.1, "Assembling an Oracle CEP Application Using Oracle CEP IDE for Eclipse"](#)
- [Section 24.2.2, "Assembling an Oracle CEP Application Manually"](#)
- [Section 24.2.3, "Assembling Applications With Foreign Stages"](#)
- [Section 24.2.4, "Assembling a Custom Adapter or Event Bean in Its Own Bundle"](#)

24.2.1 Assembling an Oracle CEP Application Using Oracle CEP IDE for Eclipse

You can use Oracle CEP IDE for Eclipse to easily assemble your Oracle CEP application.

For more information, see:

- [Section 4.5, "Exporting Oracle CEP Projects"](#)
- [Section 4.6, "Upgrading Projects"](#)

- [Section 4.7, "Managing Libraries and Other Non-Class Files in Oracle CEP Projects"](#)

If your application depends on foreign stages, see [Section 24.2.3, "Assembling Applications With Foreign Stages"](#).

24.2.2 Assembling an Oracle CEP Application Manually

Optionally, you can assemble your Oracle CEP application manually.

For simplicity, the following procedure creates a temporary directory that contains the required artifacts, and then jars up the contents of this temporary directory. This is just a suggestion and you are not required, of course, to assemble the application using this method.

Note: See the HelloWorld example source directory for a sample `build.xml` Ant file that performs many of the steps described below. The `build.xml` file is located in `ORACLE_CEP_HOME\ocep_11.1\samples\source\applications\helloworld`, where `ORACLE_CEP_HOME` refers to the main Oracle CEP installation directory, such as `d:\oracle_cep`.

To assemble an Oracle CEP application manually:

1. Open a command window and set your environment as described in "Setting Your Development Environment" in the *Oracle Complex Event Processing Getting Started*.
2. Create an empty directory, such as `output`:

```
prompt> mkdir output
```
3. Compile all application Java files into the `output` directory.
4. Create an `output/META-INF/spring` directory.
5. Copy the EPN assembly file that describes the components of your application and how they are connected into the `output/META-INF/spring` directory.
See [Section 4.3, "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.
You create these XML files during the course of creating your application, as described in [Section 1.1, "Overview of the Oracle CEP Programming Model."](#)
8. Create a `MANIFEST.MF` file that contains descriptive information about the bundle.
See [Section 24.2.2.1, "Creating the MANIFEST.MF File."](#)
9. If you need to access third-party JAR files from your Oracle CEP application, see [Section 24.2.2.2, "Accessing Third-Party JAR Files."](#)
10. Create a JAR file that contains the contents of the `output` directory.

Be sure you specify the `MANIFEST.MF` file you created in the previous step rather than the default manifest file.

You can name the JAR file anything you want. In the Oracle CEP examples, the name of the JAR file is a combination of Java package name and version, such as:

```
com.bea.wlevs.example.helloworld_1.0.0.0.jar
```

Consider using a similar naming convention to clarify which bundles are deployed to the server.

See the Apache Ant documentation at

<http://ant.apache.org/manual/CoreTasks/jar.html> for information on using the `jar` task or the Java SE documentation at

<http://java.sun.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 24.2.3, "Assembling Applications With Foreign Stages"](#).

24.2.2.1 Creating the MANIFEST.MF File

The structure and contents of the `MANIFEST.MF` file is specified by the OSGi Framework. Although the value of many of the headers in the file is specific to your application or business, many of the headers are required by Oracle CEP.

In particular, the `MANIFEST.MF` file defines the following:

- Application name—Specified with the `Bundle-Name` header.
- Symbolic application name—Specified with the `Bundle-SymbolicName` header.
Many of the Oracle CEP tools, such as the `wlevs.Admin` utility and JMX subsystem, use the symbolic name of the bundle when referring to the application.
- Application version—Specified with the `Bundle-Version` header.
- Imported packages—Specified with the `Import-Package` header.

Oracle CEP requires that you import the following packages at a minimum:

`Import-Package:`

```
com.bea.wlevs.adapter.defaultprovider;version="11.1.1.4_0",
com.bea.wlevs.ede;version="11.1.1.4_0",
com.bea.wlevs.ede.api;version="11.1.1.4_0",
com.bea.wlevs.ede.impl;version="11.1.1.4_0",
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="11.1.1.4_0",
com.bea.wlevs.util;version="11.1.1.4_0",
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="11.1.1.4_0",
```

```
com.bea.wlevs.configuration.application;version="11.1.1.4_0",
com.sun.xml.bind.v2;version="2.0.2"
```

- **Exported packages**—Specified with the `Export-Package` header. You should specify this header only if you need to share one or more application classes with other deployed applications. A typical example is sharing an event type JavaBean.

If possible, you should export packages that include only the interfaces, and not the implementation classes themselves. If other applications are using the exported classes, you will be unable to fully undeploy the application that is exporting the classes.

Exported packages are server-wide, so be sure their names are unique across the server.

The following complete `MANIFEST.MF` file is from the HelloWorld example, which extends the configuration of its adapter:

```
Manifest-Version: 1.0
Archiver-Version:
Build-Jdk: 1.6.0_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="11.1.1.4_0",
com.bea.wlevs.ede;version="11.1.1.4_0",
com.bea.wlevs.ede.impl;version="11.1.1.4_0",
com.bea.wlevs.ede.api;version="11.1.1.4_0",
org.osgi.framework;version="1.3.0",
org.apache.commons.logging;version="1.1.0",
com.bea.wlevs.spring;version="11.1.1.4_0",
com.bea.wlevs.util;version="11.1.1.4_0",
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="11.1.1.4_0",
com.bea.wlevs.configuration.application;version="11.1.1.4_0",
com.sun.xml.bind.v2;version="2.0.2"
Bundle-Name: example.helloworld
Bundle-Description: WLEvS example helloworld
Bundle-SymbolicName: helloworld
```

24.2.2.2 Accessing Third-Party JAR Files

When creating your Oracle CEP applications, you might need to access legacy libraries within existing third-party JAR files. You can ensure access to this legacy code using any of the following approaches:

- [Section 24.1.3, "Application Libraries"](#)
- [Section 24.2.2.2.1, "Accessing Third-Party JAR Files Using Bundle-Classpath"](#)
- [Section 24.2.2.2.2, "Accessing Third-Party JAR Files Using -Xbootclasspath"](#)

24.2.2.2.1 Accessing Third-Party JAR Files Using Bundle-Classpath The recommended approach is to package the third-party JAR files in your Oracle CEP application JAR file. You can put the JAR files anywhere you want.

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 CEP server application library approach instead. For more information, see [Section 24.1.3, "Application Libraries"](#).

However, to ensure that your Oracle CEP application finds the classes in the third-party JAR file, you must update the application classpath by adding the `Bundle-Classpath` header to the `MANIFEST.MF` file. Set `Bundle-Classpath` to a comma-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 4.7.1, "How to Add a Standard JAR File to an Oracle CEP Project"](#).

24.2.2.2.2 Accessing Third-Party JAR Files Using -Xbootclasspath If the JAR files include libraries used by *all* applications deployed to Oracle CEP, such as JDBC drivers, you can add the JAR file to the server's boot classpath by specifying the `-Xbootclasspath/a` option to the `java` command in the scripts used to start up an instance of the server.

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 CEP server application library approach instead. For more information, see [Section 24.1.3, "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 CEP installation directory, such as `d:\oracle_cep`.

Update the start script by adding the `-Xbootclasspath/a` option to the `java` command that executes the `wlevs_2.0.jar` file. Set the `-Xbootclasspath/a` option to the full pathname of the third-party JAR files you want to access system-wide.

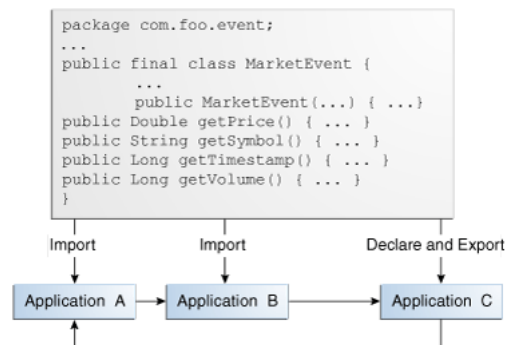
For example, if you want all deployed applications to be able to access a JAR file called `e:\jars\myExcitingJAR.jar`, update the `java` command in the start script as follows. The updated section is shown in bold (in practice, the command should be on one line):

```
%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
```

24.2.3 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 24–1](#) shows.

Figure 24–1 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 1.1.1.9, "Foreign Stages"](#)
- [Section 24.5, "Deploying Oracle CEP Applications"](#)

24.2.4 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 24.2.4.1, "How to Assemble a Custom Adapter in its Own Bundle"](#)
- [Section 24.2.4.2, "How to Assemble a Custom Event Bean in its Own Bundle"](#)

24.2.4.1 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 14.4.1, "Registering the Custom Adapter Factory"](#) describes.
 - If you are also including the event type in the bundle, register it as [Section 2.8, "Sharing Event Types Between Application Bundles"](#) describes.
 - Do *not* declare the custom adapter component using the `wl: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 14.5, "Configuring the Custom Adapter 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 4.7.4, "How to Export a Package"](#) describes.
3. Assemble and deploy the `GlobalAdapter` bundle as [Section 24.5, "Deploying Oracle CEP 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 14.4.2, "Declaring the Custom Adapter Components in your Application"](#) 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 24.2.2.1, "Creating the MANIFEST.MF File"](#) describes.

24.2.4.2 How to Assemble a Custom 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 15.3.1, "Registering the Custom Event Bean Factory"](#) describes.
- Do *not* declare the custom event bean 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 event bean, follow the usual procedure as [Section 15.4, "Configuring the Custom Event Bean Component Configuration File"](#) describes.
3. Assemble and deploy the `GlobalEventBean` bundle as [Section 24.5, "Deploying Oracle CEP 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 15.3.2, "Declaring the Custom Event Bean Components in your Application"](#) 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 24.2.2.1, "Creating the MANIFEST.MF File"](#) describes.

24.3 Managing Application Libraries

The Oracle CEP 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 CEP server start up time.

This section describes how to manage an Oracle CEP server application library, including:

- [Section 24.3.1, "How to Define the Application Library Directory Using Oracle CEP IDE for Eclipse"](#)
- [Section 24.3.2, "How to Create an Application Library Using bundler.sh"](#)
- [Section 24.3.3, "How to Create an Application Library Using Oracle CEP IDE for Eclipse"](#)
- [Section 24.3.4, "How to Update an Application Library Using Oracle CEP IDE for Eclipse"](#)
- [Section 24.3.5, "How to View an Application Library Using the Oracle CEP Visualizer"](#)

For more information, see [Section 24.1.3, "Application Libraries"](#).

24.3.1 How to Define the Application Library Directory Using Oracle CEP IDE for Eclipse

Before you can use the Oracle CEP server application library, you must update your Oracle CEP IDE for Eclipse design time configuration with the location of the application library directory.

For information on default application library configuration, see:

- [Section 24.1.3.1, "Library Directory"](#)
- [Section 24.1.3.2, "Library Extensions Directory"](#)

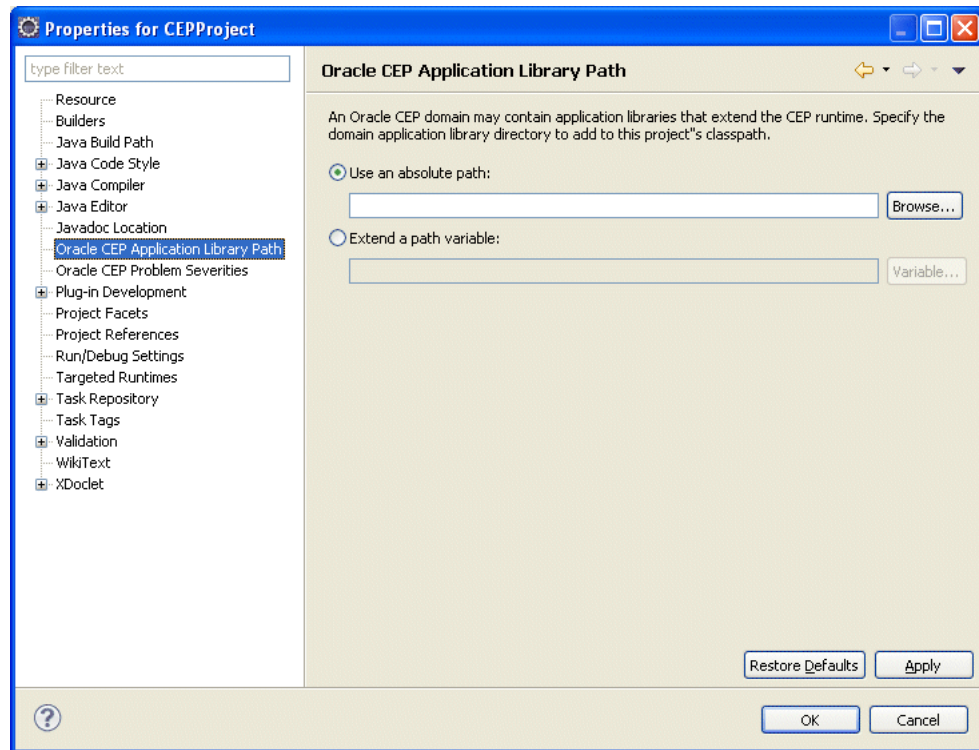
For more information, see [Section 24.3, "Managing Application Libraries"](#).

To define an application library directory using Oracle CEP IDE for Eclipse:

1. Launch the Oracle CEP IDE for Eclipse.
2. Right-click the project and select **Properties**.

The Preferences dialog appears as shown in [Figure 24-2](#).

Figure 24–2 Preferences Dialog: Application Library Path



3. Select **Oracle CEP Application Library Path**.
4. Specify the application library path as [Table 24–1](#) describes.

Table 24–1 Oracle CEP Application Library Path

Option	Description
Use an absolute path	Select this option to specify an absolute file path to the application library directory. See Section 24.3.1.1, "How to Configure an Absolute Path" .
Extend a path variable	Select this option to specify an application library path based on a path variable. See Section 24.3.1.2, "How to Extend a Path Variable" .

24.3.1.1 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 24.3.1.2, "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 CEP server domain. For more information, see [Section 5.2, "Creating Oracle CEP Servers"](#).

3. Click **OK**.

4. Click **Apply**.
5. Click **OK**.

24.3.1.2 How to Extend a Path Variable

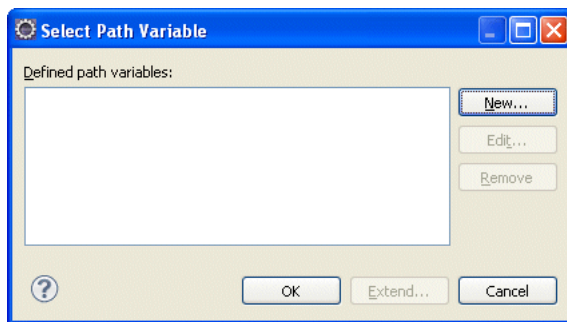
You can specify the application library path by extending a path variable. This is the most flexible approach and is appropriate for team environments. Alternatively, you can specify the application library with an absolute path as [Section 24.3.1.1, "How to Configure an Absolute Path"](#) describes.

To extend a path variable:

1. Click the **Variable** button.

The Select Path Variable dialog appears as [Figure 24–3](#) shows.

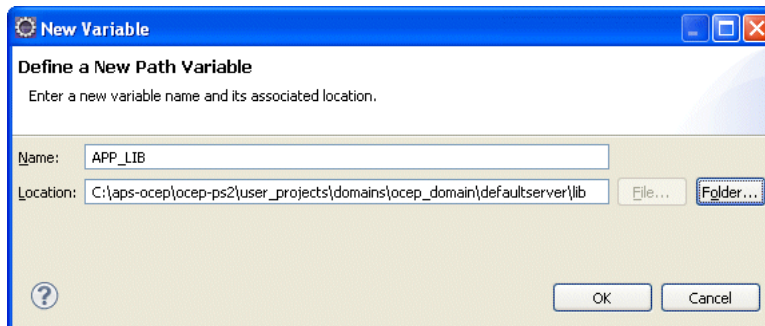
Figure 24–3 Select Path Variable Dialog



2. Click **New**.

The New Variable dialog appears as [Figure 24–4](#) shows.

Figure 24–4 New Variable Dialog



3. Configure the New Variable dialog as [Table 24–2](#) describes.

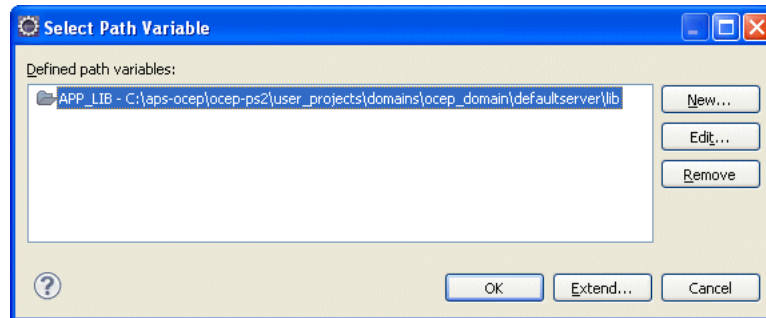
Table 24–2 Oracle CEP Application Library Path Variable

Option	Description
Name	Enter a name for the variable.
Location	Click the F older button to open a file system browser and choose the root directory to use as the application library directory. NOTE: The directory must reside within an Oracle CEP server domain. For more information, see Section 5.2, "Creating Oracle CEP Servers"

4. Click **OK**.

The new variable appears in the Select Path Variable dialog as [Figure 24–5](#) shows.

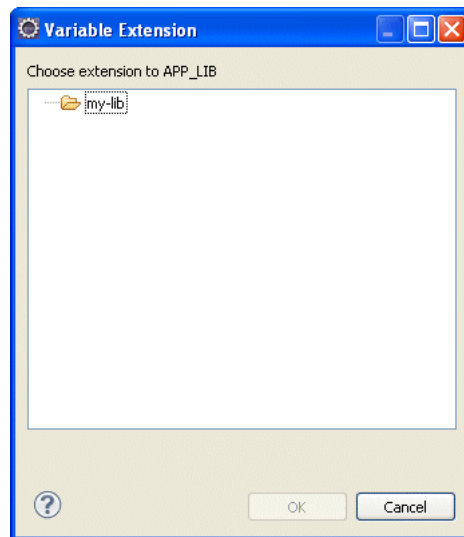
Figure 24–5 Select Path Variable: With Variable



5. Optionally, select the variable and click **Extend**.

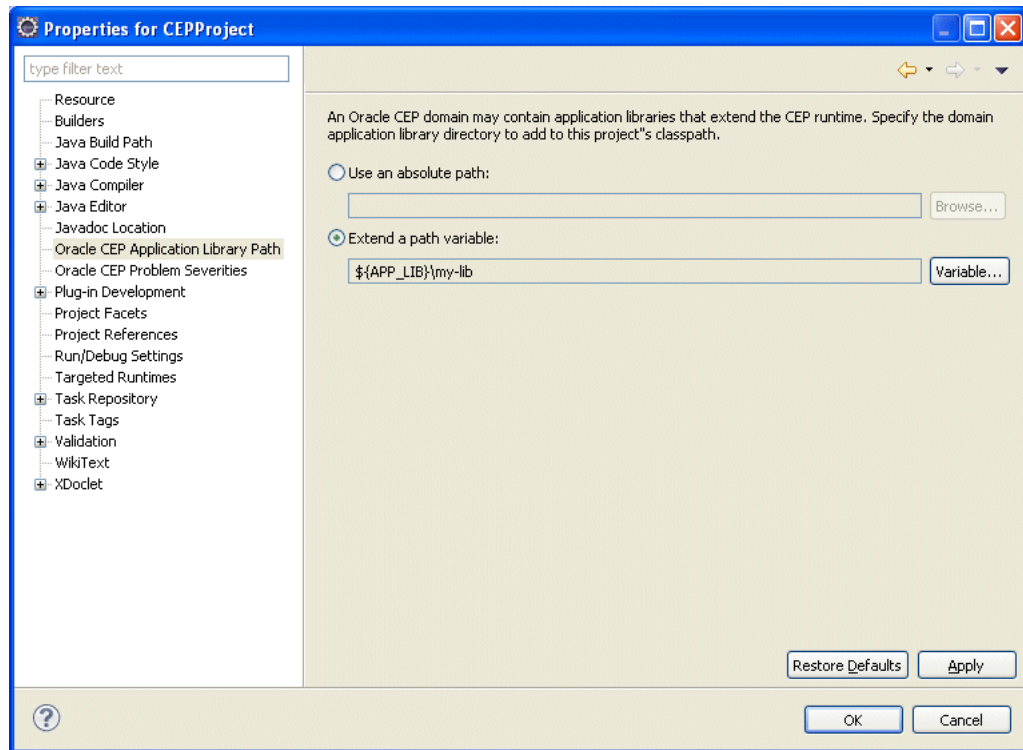
The Variable Extension dialog appears as [Figure 24–6](#) shows. This dialog shows any directories below the root directory you specified for this variable.

Figure 24–6 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 24–7](#) shows.

Figure 24–7 Preferences Dialog: Application Library Path With Path Variable

7. Click **Apply**.
8. Click **OK**.

24.3.2 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 24.3.3, "How to Create an Application Library Using Oracle CEP 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 Complex Event Processing Administrator's Guide*.

For more information, see [Section 24.1.3.3, "Creating Application Libraries"](#).

To create an application library using `bundler.sh`:

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

[Example 24–1](#) lists the `bundler.sh` command line options and [Table 24–3](#) describes them.

Example 24–1 *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 24–3 *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.

Table 24–3 (Cont.) bundler.sh Command Line Options

Argument	Description
-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 24–2 shows how to use the `bundler.sh` to create an OSGi bundle for an Oracle JDBC driver.

Example 24–2 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 24–3 shows.

Example 24–3 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 24–4 shows.

Table 24–4 Factory Class and Service Interface

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 24–4 shows the Oracle CEP server log messages showing the registration of the services.

Example 24–4 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 CEP server application library directory:

- a. If your bundle is a driver, you must put it in the library extensions directory.
See [Section 24.1.3.2, "Library Extensions Directory"](#).
- b. If your bundle is not a driver, you may put it in the library directory.
See [Section 24.1.3.1, "Library Directory"](#)

For more information, see [Section 24.1.3, "Application Libraries"](#).

4. Stop and start the Oracle CEP server.

See "Starting and Stopping Oracle CEP Servers" in the *Oracle Complex Event Processing Administrator's Guide*.

24.3.3 How to Create an Application Library Using Oracle CEP IDE for Eclipse

This procedure describes how to create an OSGi bundle for your driver using the Oracle CEP IDE for Eclipse and deploy it on the Oracle CEP server.

This is the preferred method. If do not wish to manually configure the activator implementation, see [Section 24.3.2, "How to Create an Application Library Using bundler.sh"](#).

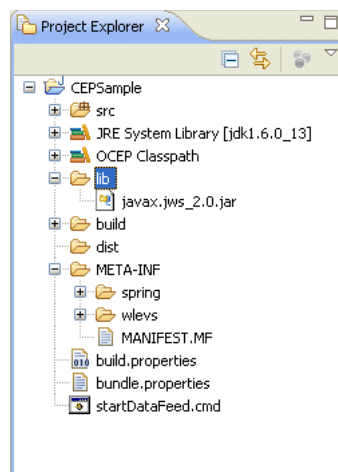
If you are creating an application library for a new JDBC driver, see "How to Access a Database Driver Using an Application Library Built With Oracle CEP IDE for Eclipse" in the *Oracle Complex Event Processing Administrator's Guide*.

To create an application library using Oracle CEP IDE for Eclipse:

1. Using the Oracle CEP IDE for Eclipse, create a new Oracle CEP project.
For more information, [Section 4.2, "Creating Oracle CEP 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 CEP IDE for Eclipse, copy your JDBC JAR file into the `lib` folder.
5. Inside the Oracle CEP IDE for Eclipse, right-click the `lib` folder and select **Refresh**.

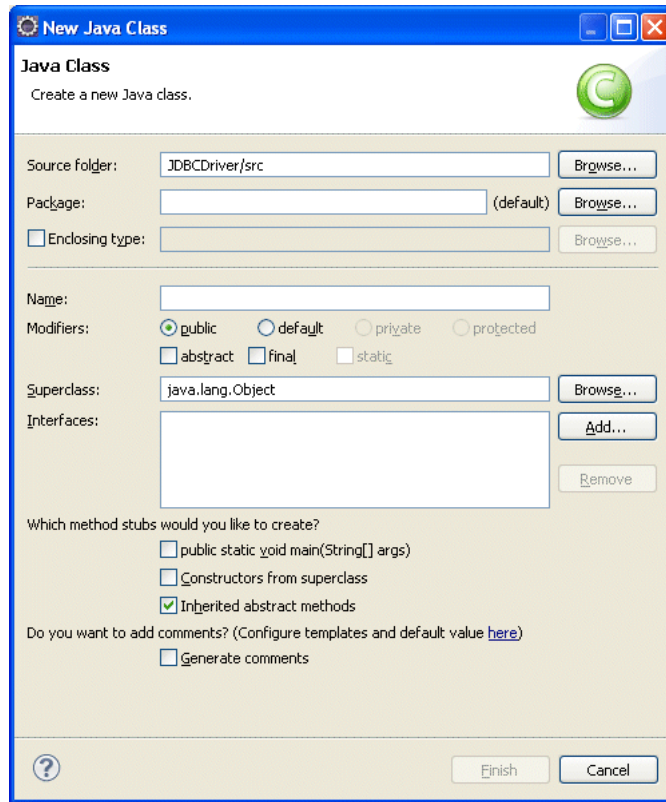
The JAR file appears in the `lib` folder as [Figure 24–8](#) shows.

Figure 24–8 Oracle CEP IDE for Eclipse lib Directory



6. Right-click the `src` directory and select **New > Class**.
The Java Class dialog appears as [Figure 24–9](#) shows.

Figure 24–9 New Java Class Dialog



7. Configure the New Java Class dialog as [Table 24–5](#) shows.

Table 24–5 New Java Class Parameters

Parameter	Description
Package	The package name. For example, <code>com.foo</code> .
Name	The name of the class. For example, <code>MyActivator</code> .

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 24–5](#) 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 24–5 MyActivator Class Implementation

```

package com.foo;

import java.util.Dictionary;
import java.util.Properties;

import javax.sql.XDataSource;
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.OracleXDataSource", "oracle.jdbc.OracleDriver"};
    private String[] interfaces= {"javax.sql.XDataSource", "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);
        }
    }

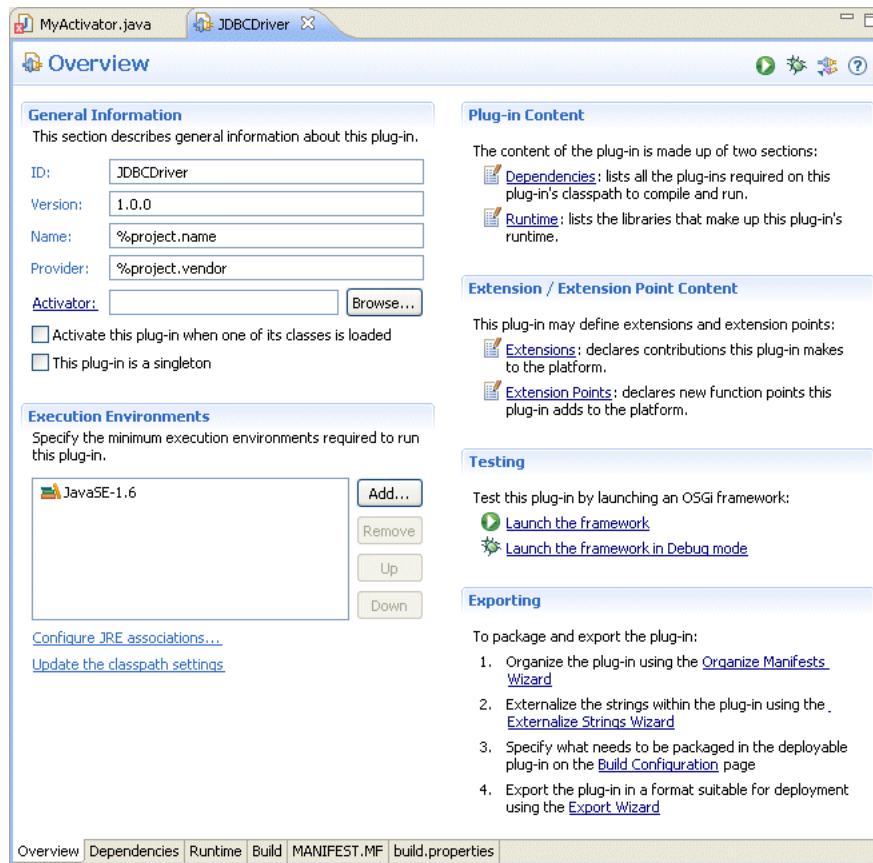
    public void stop(BundleContext bc) throws Exception {
        for (int i=0; i<serviceRegistrations.length; i++) {
            serviceRegistrations[i].unregister();
        }
    }
}

```

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

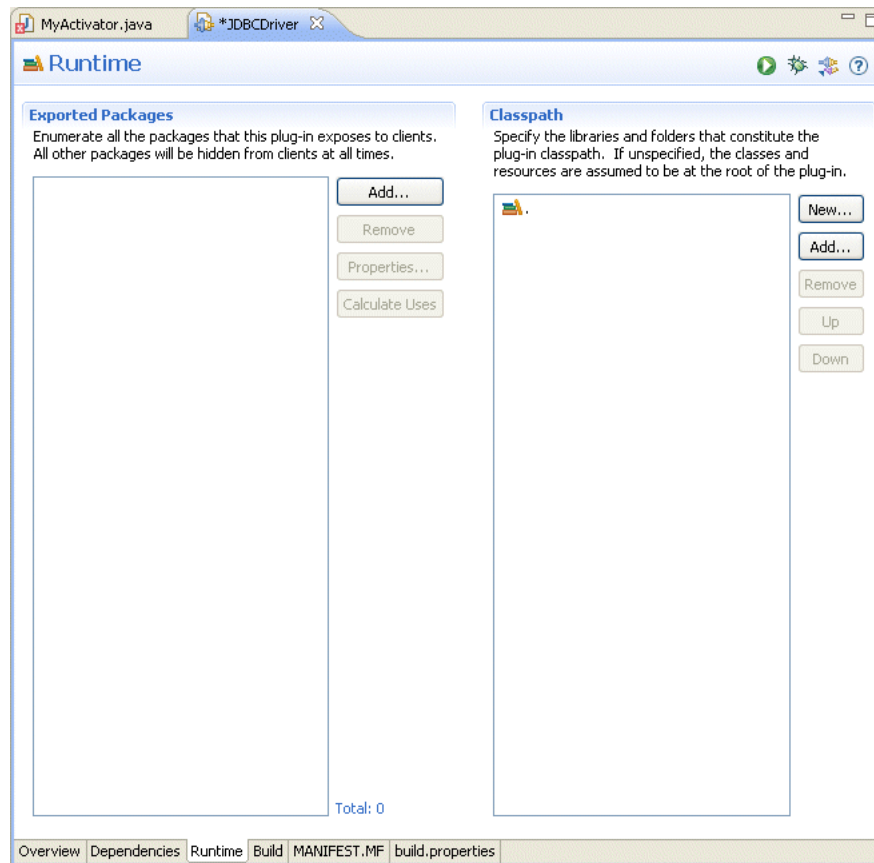
The Manifest Editor appears as [Figure 24–10](#) shows.

Figure 24–10 Manifest Editor: Overview Tab



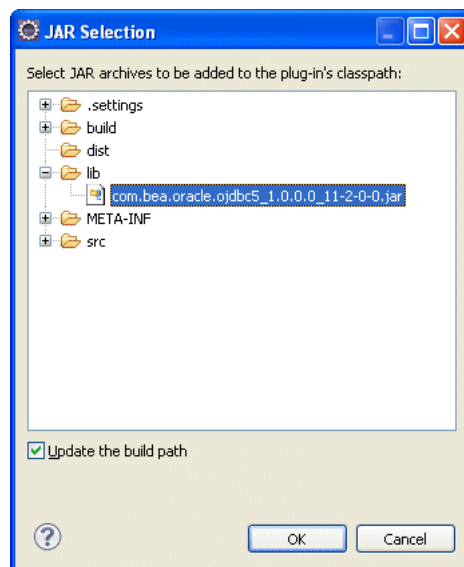
11. Click the **Runtime** tab.

The Runtime tab appears as [Figure 24–11](#) shows.

Figure 24–11 Manifest Editor: Runtime Tab

12. In the Classpath pane, click **Add**.

The JAR Selection dialog appears as [Figure 24–12](#) shows.

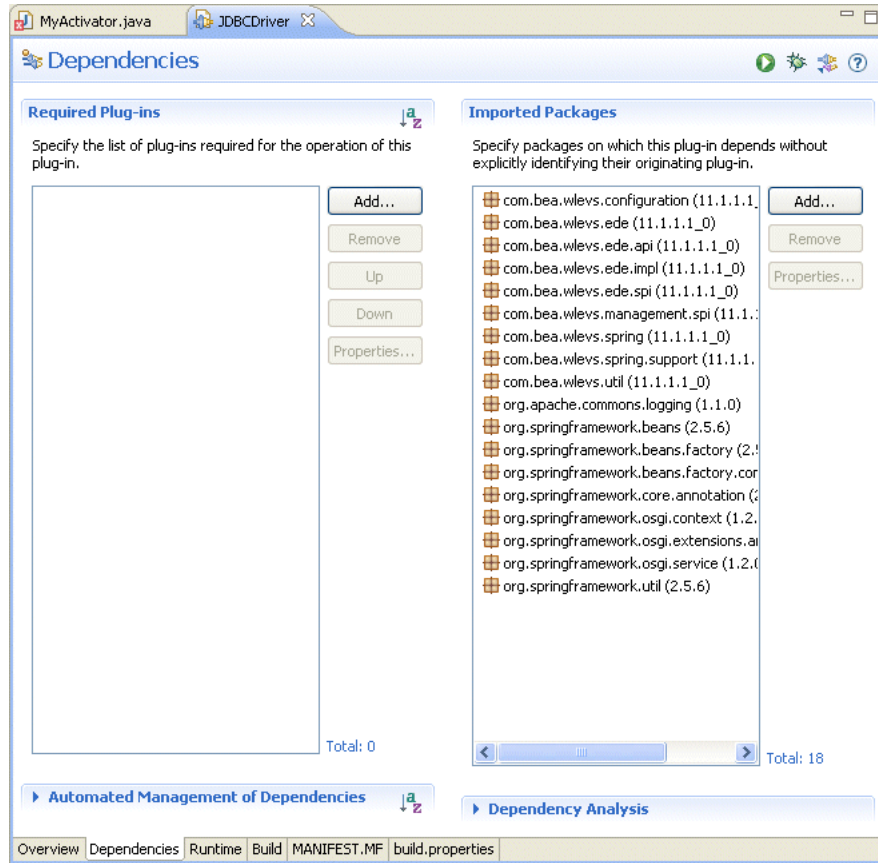
Figure 24–12 JAR Selection Dialog

13. Expand the **lib** directory and select your database driver JAR file.

14. Click **OK**.
15. Click the **Dependencies** tab.

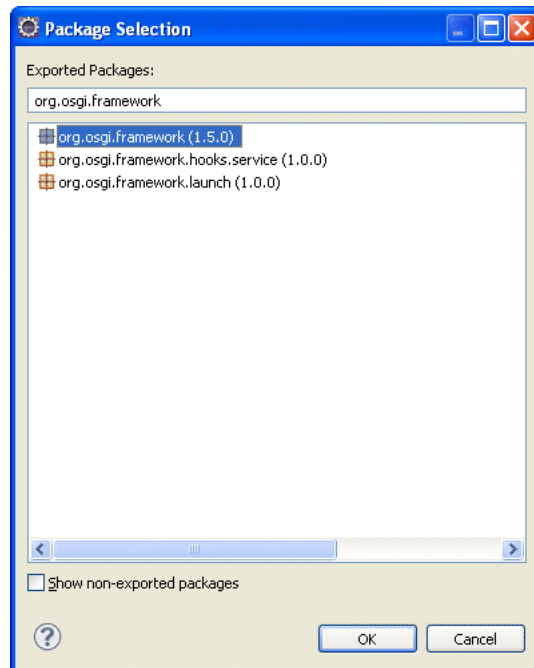
The Dependencies tab appears as [Figure 24–13](#) shows.

Figure 24–13 Manifest Editor: Dependencies Tab



16. In the Imported Packages pane, click **Add**.

The Package Selection dialog appears as [Figure 24–14](#) shows.

Figure 24–14 Package Selection Dialog

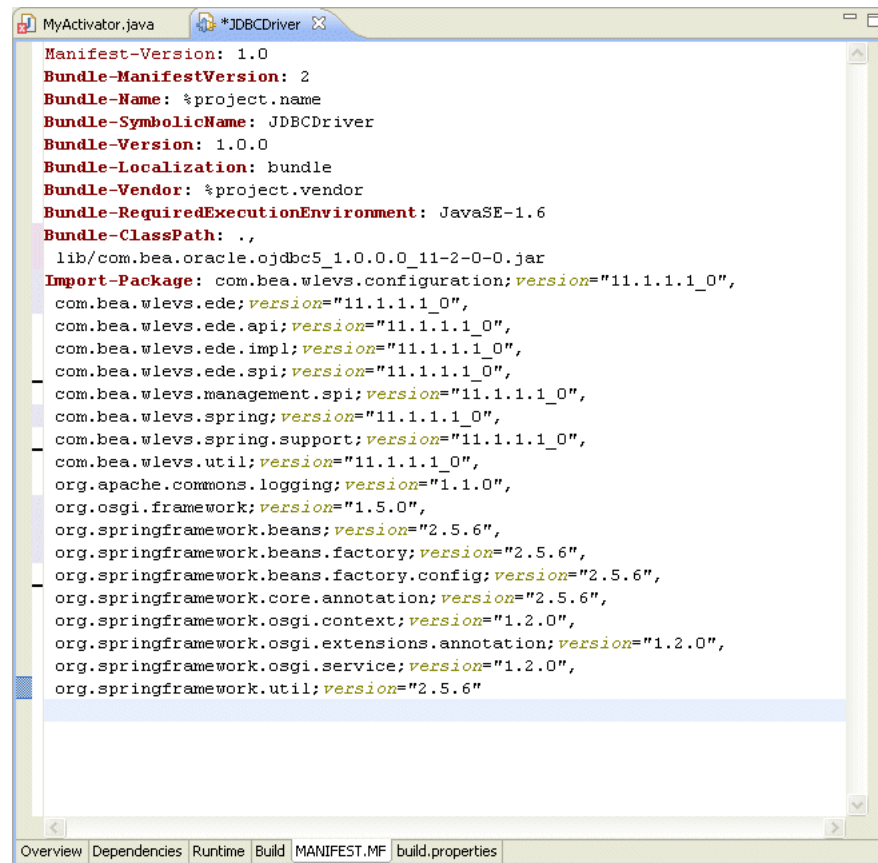
17. In the **Exported Packages** field, enter `org.osgi.framework`.

The list box shows all the packages with that prefix as [Figure 24–14](#) 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 24–15](#) shows.

Figure 24–15 Manifest Editor

20. Un-JAR your JAR to a temporary directory as [Example 24–6](#) shows.

Example 24–6 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 24–7](#) shows.

Example 24–7 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,ora
cle.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 24–8](#) shows.

The value of the `Bundle-Activator` is the fully qualified class name of your `Activator` class.

Example 24–8 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: 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, ...
...

```

- 23.** Add a `DynamicImport-Package` element to the Manifest Editor as [Example 24–9](#) shows.

Example 24–9 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: JDBCDriver
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 CEP application to a JAR file.

For more information, see [Section 4.5.1, "How to Export an Oracle CEP Project"](#).

25. Copy the bundler JAR to the appropriate Oracle CEP server application library directory:
 - a. If your bundle is a driver, you must put it in the library extensions directory. See [Section 24.1.3.2, "Library Extensions Directory"](#).
 - b. If your bundle is not a driver, you may put it in the library directory. See [Section 24.1.3.1, "Library Directory"](#)

For more information, see [Section 24.1.3, "Application Libraries"](#).

26. Stop and start the Oracle CEP server.

See "Starting and Stopping Oracle CEP Servers" in the *Oracle Complex Event Processing Administrator's Guide*.

24.3.4 How to Update an Application Library Using Oracle CEP 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 CEP server you used to create a server runtime in the Oracle CEP IDE for Eclipse.
- On the production Oracle CEP server to which you deploy dependent applications.

These changes need not be performed simultaneously: you must make the change to the local Oracle CEP server before making code changes to projects that depend on the application library change; you must make the change to the production Oracle CEP server before you deploy applications that depend on the application library change.

For more information, see [Section 24.3, "Managing Application Libraries"](#).

To update an application library using Oracle CEP IDE for Eclipse:

1. Add a new or revised bundle to the application library extension or application library directory on the production Oracle CEP 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 24.1.3.2, "Library Extensions Directory"](#)
- [Section 24.1.3.1, "Library Directory"](#)
- [Section 24.1.4, "Deployment and Deployment Order"](#)

2. Stop and start the production Oracle CEP server.

The Oracle CEP server refreshes itself from the updated application library extension or application library directory.

For more information, see:

- "Starting and Stopping an Oracle CEP Server in a Standalone-Server Domain" in the *Oracle Complex Event Processing Administrator's Guide*

- "Starting and Stopping an Oracle CEP Server in a Multi-Server Domain" in the *Oracle Complex Event Processing Administrator's Guide*
- 3. Add the same new or revised bundle to the application library extension or application library directory on your Oracle CEP IDE for Eclipse targeted runtime Oracle CEP server.
- 4. Start the Oracle CEP IDE for Eclipse.
- 5. Right click a project and select **Refresh Targeted Runtimes**.
The Oracle CEP IDE for Eclipse refreshes this project from the updated application library extension or application library directory on your Oracle CEP IDE for Eclipse targeted runtime Oracle CEP 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 24.1.2, "Application Dependencies"](#).
- 7. Assemble and deploy your application to the production Oracle CEP server.
For more information, see [Section 24.5, "Deploying Oracle CEP Applications"](#).
The dependencies you defined for this application in the Oracle CEP IDE for Eclipse at development time will be satisfied by the components you installed in the application library of your production Oracle CEP server at run time.

24.3.5 How to View an Application Library Using the Oracle CEP Visualizer

Using the Oracle CEP Visualizer, you can view the application libraries deployed to the Oracle CEP server.

You can view libraries from both the library extensions directory and libraries directory.

Note: You cannot deploy an application library to an Oracle CEP server using the Oracle CEP Visualizer. You may only deploy Oracle CEP applications to an Oracle CEP server using the Oracle CEP Visualizer.

For more information, see:

- "How to View the Application Libraries Deployed to an Oracle CEP Server" in the *Oracle Complex Event Processing Visualizer User's Guide*
- [Section 24.1.3.2, "Library Extensions Directory"](#)
- [Section 24.1.3.1, "Library Directory"](#)
- [Section 24.3, "Managing Application Libraries"](#)

24.4 Managing Log Message Catalogs

This section describes how to manage log message catalogs that you can use to localize an Oracle CEP application, including:

- [Section 24.4.1, "Using Message Catalogs With Oracle CEP Server"](#)

- [Section 24.4.2, "How to Parse a Message Catalog to Generate Logger and TextFormatter Classes for Localization"](#)

For more information, see:

- ["Configuring Logging and Debugging for Oracle CEP" in the *Oracle Complex Event Processing Administrator's Guide*](#)
- [Appendix G, "Schema Reference: Message Catalog msgcat.dtd"](#)
- [Appendix H, "Schema Reference: Locale Message Catalog l10n_msgcat.dtd"](#)

24.4.1 Using Message Catalogs With Oracle CEP 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 24.4.1.1, "Message Catalog Hierarchy"](#)
- [Section 24.4.1.2, "Guidelines for Naming Message Catalogs"](#)
- [Section 24.4.1.3, "Using Message Arguments"](#)
- [Section 24.4.1.4, "Message Catalog Formats"](#)
- [Section 24.4.1.5, "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"](#)

24.4.1.1 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.

24.4.1.2 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.

24.4.1.3 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 24–10](#), {0} represents the file that cannot be opened, while {1} represents the file that will be opened in its place.

Example 24–10 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 24–10](#) 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.

24.4.1.4 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.

24.4.1.4.1 Log Message Catalog [Example 24–11](#) 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 24–11 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"](#).

24.4.1.4.2 Simple Text Catalog Example 24–12 shows a simple text catalog, MyUtilLabels.xml, with one simple text definition.

Example 24–12 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"](#).

24.4.1.4.3 Locale-Specific Catalog Example 24–13 shows a French translation of a message that is available in `.. \msgcat\fr\MyUtilLabels.xml`.

Example 24–13 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"](#).

24.4.1.5 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.

24.4.2 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 CEP server classpath.

Oracle recommends that the `i18n_user.properties` file reside in the Oracle CEP server classpath. If the `i18n_user.properties` file is in `targetdirectory`, then `targetdirectory` should be in the Oracle CEP server classpath.

To parse a message catalog to generate `Logger` and `TextFormatter` classes:

1. Create your message catalog XML file.
See [Section 24.4.1, "Using Message Catalogs With Oracle CEP Server"](#).
2. Set up your development environment.
See "Setting Your Development Environment" in the *Oracle Complex Event Processing Getting Started*.
3. Execute the `weblogic.i18ngen` utility using the following syntax:

```
java weblogic.i18ngen [options] [filelist]
```

Where:

- `options`: see [Table 24-6](#).
- `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 24-6 `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.
<code>-d targetdirectory</code>	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 <code>targetdirectory</code> . Files are placed in appropriate directories based on the <code>i18n_package</code> and <code>l10n_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.
<code>-n</code>	Parse and validate, but do not generate classes.

Table 24–6 (Cont.) weblogic.i18ngen Utility Options

Option	Description
-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>).
-l10n	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 <code>-d</code> 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 CEP server classpath.
6. Import the following packages in your Oracle CEP application:
 - `weblogic.i18n.logging`
 - `weblogic.logging`
7. Assemble and deploy your application, including your log message resource bundles.

24.5 Deploying Oracle CEP Applications

After you assemble your Oracle CEP application, you deploy it to an Oracle CEP server domain.

This section describes:

- [Section 24.5.1, "How to Deploy an Oracle CEP Application Using Oracle CEP IDE for Eclipse"](#)
- [Section 24.5.2, "How to Deploy an Oracle CEP Application Using Oracle CEP Visualizer"](#)
- [Section 24.5.3, "How to Deploy an Oracle CEP Application Using the Deployer Utility"](#)

For more information, see:

- "Deploying an Application to an Oracle CEP Standalone-Server Domain" in the *Oracle Complex Event Processing Administrator's Guide*
- "Deploying an Oracle CEP Application to a Multi-Server Domain" in the *Oracle Complex Event Processing Administrator's Guide*

24.5.1 How to Deploy an Oracle CEP Application Using Oracle CEP IDE for Eclipse

You can deploy an Oracle CEP application using Oracle CEP IDE for Eclipse.

Using the Oracle CEP IDE for Eclipse, you can deploy an application to 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 24.2.3, "Assembling Applications With Foreign Stages"](#) describes.

To deploy an Oracle CEP application using Oracle CEP IDE for Eclipse:

1. Assemble your Oracle CEP application.
See [Section 24.2, "Assembling an Oracle CEP Application."](#)
2. Use the Oracle CEP IDE for Eclipse to deploy your application.
See [Section 5.3.6, "How to Deploy an Application to an Oracle CEP Server"](#).

24.5.2 How to Deploy an Oracle CEP Application Using Oracle CEP Visualizer

The simplest way to deploy an Oracle CEP application to an Oracle CEP server domain is to use the Oracle CEP Visualizer.

Using the Oracle CEP Visualizer, you can deploy an application to either a stand-alone or multi-server domain.

Note: If you are using foreign stages, beware of the rules governing deployment and redeployment of dependent stages as [Section 24.2.3, "Assembling Applications With Foreign Stages"](#) describes.

To deploy an Oracle CEP application using Oracle CEP Visualizer:

1. Assemble your Oracle CEP application.
See [Section 24.2, "Assembling an Oracle CEP Application."](#)
2. Start the Oracle CEP Visualizer.
See [Section 5.3.9, "How to Start the Oracle CEP Visualizer from Oracle CEP IDE for Eclipse"](#).
3. Use the Oracle CEP Visualizer to deploy your application.
See "Deploying an Application" in the *Oracle Complex Event Processing Visualizer User's Guide*.

24.5.3 How to Deploy an Oracle CEP Application Using the Deployer Utility

The following procedure describes how to deploy an application to Oracle CEP using the Deployer command-line utility.

Using the Deployer, you can deploy an application to either a stand-alone or multi-server domain.

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

Note: If you are using foreign stages, beware of the rules governing deployment and redeployment of dependent stages as [Section 24.2.3, "Assembling Applications With Foreign Stages"](#) describes.

To deploy an Oracle CEP application using the Deployer utility:

1. Assemble your Oracle CEP application.

See [Section 24.2, "Assembling an Oracle CEP Application."](#)

2. Open a command window and set your environment as described in "Setting Your Development Environment" in the *Oracle Complex Event Processing Getting Started*.
3. Update your CLASSPATH variable to include the `wlevsdeploy.jar` JAR file, located in the `ORACLE_CEP_HOME/occep_11.1/bin` directory where, `ORACLE_CEP_HOME` refers to the main Oracle CEP installation directory, such as `/oracle_cep`.

Note: If you are running the Deployer on a remote computer, see "Running the Deployer Utility Remotely" in the *Oracle Complex Event Processing Administrator's Guide*.

4. Be sure you have configured Jetty for the Oracle CEP instance to which you are deploying your application.

For more information, see "Configuring Jetty for Oracle CEP" in the *Oracle Complex Event Processing Administrator's Guide*.

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 CEP is running.
- `port` refers to the port number to which Oracle CEP listens; default value is 9002.

This port is specified in the `DOMAIN_DIR/config/config.xml` file that describes your Oracle CEP domain, where `DOMAIN_DIR` refers to your domain directory.

The port number is the value of the `<Port>` child element of the `<Netio>` element:

```
<Netio>
  <Name>NetIO</Name>
  <Port>9002</Port>
</Netio>
```

- `user` refers to the username of the Oracle CEP administrator.
- `password` refers to the password of the Oracle CEP administrator.
- `application_jar_file` refers to your application JAR file, assembled into an OSGi bundle as described in [Section 24.2, "Assembling an Oracle CEP"](#)

Application. This file must be located on the same computer from which you execute the Deployer utility.

For example, if Oracle CEP is running on host `ariel`, listening on port `9002`, username and password of the administrator is `wlevs/wlevs`, and your application JAR file is called `myapp_1.0.0.0.jar` and is located in the `/applications` directory, then the command is (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 CEP automatically starts the application and the adapter components immediately start listening for incoming events.

The Deployer utility provides additional options to resume, suspend, update, and uninstall an application JAR file, as well as deploy an application to a specified group of a multi-server domain. For more information, see "Deployer Command-Line Reference" in the *Oracle Complex Event Processing Administrator's Guide*.

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 CEP Multi-Server Domain Administration" in the *Oracle Complex Event Processing Administrator's Guide*.

Oracle CEP uses the `deployments.xml` file to internally maintain its list of deployed application OSGi bundles. This file is located in the `DOMAIN_DIR/servername` directory, where `DOMAIN_DIR` refers to the main domain directory corresponding to the server instance to which you are deploying your application and `servername` refers to the actual server. See [Appendix B.3, "Deployment Schema deployment.xsd"](#) for information about this file. This information is provided for your information only; Oracle does not recommend updating the `deployments.xml` file manually.

Testing Applications With the Load Generator and csvgen Adapter

This section contains information on the following subjects:

- [Section 25.1, "Overview of Testing Applications With the Load Generator and csvgen Adapter"](#)
- [Section 25.2, "Configuring and Running the Load Generator Utility"](#)
- [Section 25.3, "Creating a Load Generator Property File"](#)
- [Section 25.4, "Creating a Data Feed File"](#)
- [Section 25.5, "Configuring the csvgen Adapter in Your Application"](#)

25.1 Overview of Testing Applications With the Load Generator and csvgen Adapter

The load generator is a simple utility provided by Oracle CEP to simulate a data feed. The utility is useful for testing the Oracle CQL or EPL rules in your application without needing to connect to a real-world data feed.

The load generator reads an ASCII file that contains the sample data feed information and sends each data item to the configured port. The load generator reads items from the sample data file in order and inserts them into the channel, looping around to the beginning of the data file when it reaches the end; this ensures that a continuous stream of data is available, regardless of the number of data items in the file. You can configure the rate of sent data, from the rate at which it starts, the final rate, and how long it takes the load generator to ramp up to the final rate.

In your application, you must use the Oracle CEP-provided `csvgen` adapter, rather than your own adapter, to read the incoming data; this is because the `csvgen` adapter is specifically coded to decipher the data packets generated by the load generator.

If you redeploy your application, you must also restart the load generator.

For more information on testing and debugging, see [Section 5.4, "Debugging an Oracle CEP Application Running on an Oracle CEP Server"](#).

25.2 Configuring and Running the Load Generator Utility

This procedure describes how to configure and run the load generator utility.

To configure and run the load generator utility:

1. Optionally create a property file that contains configuration properties for particular run of the load generator; these properties specify the location of the file that contains simulated data, the port to which the generator feeds the data, and so on.

Oracle CEP provides a default property file you can use if the default property values are adequate.

See [Section 25.3, "Creating a Load Generator Property File."](#)

2. Create a file that contains the actual data feed values.

See [Section 25.4, "Creating a Data Feed File."](#)

3. Configure the `csvgen` adapter so that it correctly reads the data feed generated by the load generator. You configure the adapter in the EPN assembly file that describes your Oracle CEP application.

See [Section 25.5, "Configuring the csvgen Adapter in Your Application."](#)

4. Be sure that you configure a builder factory for creating your event types. Although specifying event type builder factories is typically an optional task, it is required when using the load generator.

See [Chapter 2.6, "Using an Event Type Builder Factory"](#) for details.

5. Open a command window and set your environment as described in "Setting Your Development Environment" in the *Oracle Complex Event Processing Getting Started*.

6. Change to the `ORACLE_CEP_HOME\ocep_11.1\utils\load-generator` directory, where `ORACLE_CEP_HOME` refers to the main Oracle CEP installation directory, such as `d:\oracle_cep`.

7. Run the load generator specifying the properties file you created in step 1 to begin the simulated data feed. For example, if the name of your properties file is `c:\loadgen\myDataFeed.prop`, execute the following command:

```
prompt> runloadgen.cmd c:\loadgen\myDataFeed.prop
```

25.3 Creating a Load Generator Property File

The load generator uses an ASCII properties file for its configuration purposes. Properties include the location of the file that contains the sample data feed values, the port to which the utility should send the data feed, and so on.

Oracle CEP provides a default properties file called `csvgen.prop`, located in the `ORACLE_CEP_HOME\ocep_11.1\utils\load-generator` directory, where `ORACLE_CEP_HOME` refers to the main Oracle CEP installation directory, such as `d:\oracle_cep`.

The format of the file is simple: each property-value pair is on its own line. The following example shows the default `csvgen.prop` file; Oracle recommends you use this file as a template for your own property file:

```
test.csvDataFile=test.csv
test.port=9001
test.packetType=CSV
test.mode=client
test.senders=1
test.latencyStats=false
test.statInterval=2000
```

Caution: If you create your own properties file, you must include the `test.packetType`, `test.mode`, `test.senders`, `test.latencyStats`, and `test.statInterval` properties exactly as shown above.

In the preceding sample properties file, the file that contains the sample data is called `test.csv` and is located in the same directory as the properties file. The load generator will send the data feed to port 9001.

The following table lists the additional properties you can set in your properties file.

Table 25–1 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 Section 25.5, "Configuring the csvgen Adapter in Your Application" 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

25.4 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 CEP application.

[Example 25–1](#) shows an `EmployeeEvent` and [Example 25–2](#) shows a load generator data feed file corresponding to this event type.

Example 25–1 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 25–2 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 25-1](#) 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 ["Event Types for use With the csvgen Adapter"](#) on page 2-5.

25.5 Configuring the csvgen Adapter in Your Application

When using the load generator utility, you must use the `csvgen` adapter in your application because this Oracle CEP-provided adapter is specifically coded to read the data packets generated by the load generator.

You register the `csvgen` adapter using the `wlevs:adapter` element in the EPN assembly file of your application, as with all adapters. Set the `provide` attribute to `csvgen` to specify that the provider is the `csvgen` adapter, rather than your own adapter. Additionally, you must specify the following child tags:

- `wlevs:instance-property` element with name attribute `port` and value attribute `configured_port`, where `configured_port` corresponds to the value of the `test.port` property in the load generator property file. See [Section 25.3, "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:meta` 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 25.4, "Creating a Data Feed File."](#)

To configure the csvgen adapter that receives this data, use the following wlevs:adapter element:

```
<wlevs:adapter id="csvgenAdapter" provider="csvgen">
  <wlevs:instance-property name="port" value="9001"/>
  <wlevs:instance-property name="eventName" value="PersonType" />
  <wlevs:instance-property name="eventPropertyNames" value="name,age,birthplace" />
</wlevs:adapter>
```

Note how the bolded values in the adapter configuration example correspond to the PersonType event type registration.

If you use the wlevs:class element to specify your own JavaBean when registering the event type, then the eventPropertyNames value corresponds to the JavaBean properties. For example, if your JavaBean has a getName method, then one of the properties of your JavaBean is name.

For more information on event types, see [Chapter 2.1.4, "Creating Oracle CEP 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 the Event Processing Network (EPN), including:

- [Section 26.1, "Overview of Testing Applications With the Event Inspector"](#)
- [Section 26.2, "Configuring the Event Inspector HTTP Pub-Sub Server"](#)
- [Section 26.3, "Injecting Events"](#)
- [Section 26.4, "Tracing Events"](#)

Note: The Event Inspector service is not for use on a production Oracle CEP server. It is for use only during development.

26.1 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 26.1.1, "Tracing Events"](#)
- [Section 26.1.2, "Injecting Events"](#)
- [Section 26.1.3, "Event Inspector Event Types"](#)
- [Section 26.1.4, "Event Inspector HTTP Publish-Subscribe Channel and Server"](#)
- [Section 26.1.5, "Event Inspector Clients"](#)

For more information on testing and debugging, see [Section 5.4, "Debugging an Oracle CEP Application Running on an Oracle CEP Server"](#).

26.1.1 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 26.1.3, "Event Inspector Event Types"](#)
- [Section 26.1.4, "Event Inspector HTTP Publish-Subscribe Channel and Server"](#)
- [Section 26.4, "Tracing Events"](#)

26.1.2 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 26.1.3, "Event Inspector Event Types"](#)
- [Section 26.1.4, "Event Inspector HTTP Publish-Subscribe Channel and Server"](#)
- [Section 26.3, "Injecting Events"](#)

26.1.3 Event Inspector Event Types

All Oracle CEP event types are supported: `JavaBean`, `Java Class`, `MapEvent`, 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 26–1](#) shows. [Table 26–1](#) lists the required attributes.

Example 26–1 Event Inspector JSON Event

```

{
  "event-type": "myEventType",
  "operation": "insert",
  "binding": "outbound",
  "value":{
    "firstname": "Jane",
    "lastname": "Doe",
    "phone": {
      "code": 12345,
      "number": "office"
    },
  },
}

```

Table 26–1 Event Inspector JSON Event Required Attributes

Attribute	Description
event-type	The name of the Oracle CEP 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"> ▪ insert: insert event. ▪ delete: delete event ▪ update: update event ▪ heartbeat: heartbeat event
binding	One of: <ul style="list-style-type: none"> ▪ inbound: injected event. ▪ outbound: trace event.
value	One or more JSON-formatted event properties as defined by the event-type.

For more information, see:

- <http://www.json.org/>
- Section 1.1.2, "Oracle CEP Event Types"
- Section 26.1.1, "Tracing Events"
- Section 26.1.2, "Injecting Events"

26.1.4 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 CEP server on which the Oracle CEP EPN stage is running.
- *APPLICATIONNAME*: the name of the Oracle CEP 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-service` 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 26.2.1, "How to Configure a Local Event Inspector HTTP Pub-Sub Server"](#).
- **Remote:** you configure your `config.xml` file with an `event-inspector-service` 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 26.2.2, "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 26.1.1, "Tracing Events"](#)
- [Section 26.1.2, "Injecting Events"](#)

26.1.5 Event Inspector Clients

The Event Inspector service supports the following clients:

- [Section 26.1.5.1, "Oracle CEP Visualizer"](#)

26.1.5.1 Oracle CEP Visualizer

You can access the Event Inspector service using the Oracle CEP Visualizer.

For more information, see "Testing Applications With the Event Inspector" in the *Oracle Complex Event Processing Visualizer User's Guide*.

26.2 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 26.2.1, "How to Configure a Local Event Inspector HTTP Pub-Sub Server"](#)
- [Section 26.2.2, "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 1.1.5, "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 8, "Configuring HTTP Publish-Subscribe Server Adapters"](#).

26.2.1 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 26.2.2, "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 CEP IDE for Eclipse.
See [Section 6.1, "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-service` element as [Example 26–2](#) shows.

Example 26–2 Event Inspector Service Local HTTP Pub-Sub Server

```
<event-inspector-service>
  <name>myEventInspectorConfig</name>
  <pubsub-server-name>myPubSub</pubsub-server-name>
</event-inspector-service>
```

Where the `pubsub-server-name` value `myPubSub` is the value of the `http-pubsub` element name child element as defined in the local Oracle CEP server `config.xml` file as [Example 26–3](#) shows.

Example 26–3 Oracle CEP 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.

26.2.2 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 26.2.1, "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 CEP IDE for Eclipse.
See [Section 6.1, "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-service` element as [Example 26-4](#) shows.

Example 26-4 Event Inspector Service Remote HTTP Pub-Sub Server

```
<event-inspector-service>
  <name>myEventInspectorConfig</name>
  <pubsub-server-url>http://HOST:PORT/PATH</pubsub-server-url>
</event-inspector-service>
```

Where:

- *HOST*: is the host name or IP address of the remote Oracle CEP server.
- *PORT*: the remote Oracle CEP server `netio` port as defined in the remote Oracle CEP server `config.xml` file. Default: 9002.
- *PATH*: the value of the `http-pubsub` element `path` child element as defined in the remote Oracle CEP server `config.xml` file.

Given the `http-pubsub` configuration that [Example 26-3](#) shows, a valid `pubsub-server-url` would be:

```
http://remotehost:9002/pubsub
```

Example 26-5 Oracle CEP 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.

26.3 Injecting Events

After you configure the Event Inspector service HTTP pub-sub server, you can use Event Inspector clients to inject events.

This section describes how to use Event Inspector service clients to inject events, including:

- [Section 26.3.1, "How to Inject Events Using Oracle CEP Visualizer"](#)

For more information, see:

- [Section 26.3, "Injecting Events"](#)
- [Section 26.2, "Configuring the Event Inspector HTTP Pub-Sub Server"](#)

26.3.1 How to Inject Events Using Oracle CEP Visualizer

You can use the Oracle CEP Visualizer to inject events into any stage of your EPN.

For more information, see:

- "How to Inject a Simple Event on an Event Inspector Service Dynamic Channel" in the *Oracle Complex Event Processing Visualizer User's Guide*
- "How to Inject an Event as a JSON String on an Event Inspector Service Dynamic Channel" in the *Oracle Complex Event Processing Visualizer User's Guide*

26.4 Tracing Events

After you configure the Event Inspector service HTTP pub-sub server, you can use Event Inspector clients to trace events.

This section describes how to use Event Inspector service clients to trace events, including:

- [Section 26.4.1, "How to Trace Events Using Oracle CEP Visualizer"](#)

For more information, see:

- [Section 26.4, "Tracing Events"](#)
- [Section 26.2, "Configuring the Event Inspector HTTP Pub-Sub Server"](#)

26.4.1 How to Trace Events Using Oracle CEP Visualizer

You can use the Oracle CEP Visualizer to trace events flowing out of any stage of your EPN.

For more information, see:

- "How to Trace Events on a Dynamic Channel" in the *Oracle Complex Event Processing Visualizer User's Guide*

Performance Tuning

This chapter describes techniques for increasing Oracle CEP server and application performance, including:

- [Section 27.1, "EPN Performance Tuning"](#)
- [Section 27.2, "High Availability Performance Tuning"](#)

27.1 EPN Performance Tuning

When creating your EPN, consider the following performance tuning options:

- [Section 27.1.1, "Event Partitioner Channel"](#)
- [Section 27.1.2, "Batching Channel"](#)
- [Section 27.1.3, "Scalability Using the ActiveActiveGroupBean"](#)

For more information, see [Chapter 22, "Understanding Scalability"](#).

27.1.1 Event Partitioner Channel

You can improve scalability by configuring an event partitioner channel. An event partitioner provides a mechanism 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 22.2.1, "EventPartitioner"](#).

27.1.2 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 9.1.6, "Batch Processing Channels"](#).

27.1.3 Scalability Using the ActiveActiveGroupBean

Using the `com.oracle.cep.cluster.hagroups.ActiveActiveGroupBean`, you can partition an incoming JMS stream in Oracle CEP applications by utilizing the notification groups that the `ActiveActiveGroupBean` creates.

For more information, see [Section 22.2.2, "ActiveActiveGroupBean"](#).

27.2 High Availability Performance Tuning

When creating high-availability applications for deployment to multi-server domains, consider the following performance tuning options:

- [Section 27.2.1, "Host Configuration"](#)
- [Section 27.2.2, "High Availability Input Adapter and Quality of Service"](#)
- [Section 27.2.3, "High Availability Input Adapter Configuration"](#)
- [Section 27.2.4, "Broadcast Output Adapter Configuration"](#)
- [Section 27.2.5, "Oracle Coherence Performance Tuning Options"](#)

For more information, see [Section 20.3, "Designing an Oracle CEP Application for High Availability"](#)

27.2.1 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.

27.2.2 High Availability Input Adapter and Quality of Service

The Oracle CEP 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 20.2.1, "Simple Failover"](#) and [Section 20.2.2, "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.

27.2.3 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 21-3, "Child Elements of ha-inbound-adapter for the High Availability Input Adapter"](#).

27.2.4 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 21–9, "Child Elements of ha-broadcast-adapter for the Broadcast Output Adapter"](#).

27.2.5 Oracle Coherence Performance Tuning Options

When configuring Oracle Coherence in a high-availability architecture, consider the following performance tuning options:

- [Section 27.2.5.1, "Oracle Coherence Heartbeat Frequency"](#)
- [Section 27.2.5.2, "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.

27.2.5.1 Oracle Coherence Heartbeat Frequency

To reduce failover time, increase Coherence heartbeat timeout machine frequency and reduce the number of heartbeats before failure.

27.2.5.2 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 VIII

Oracle CEP Reference

Part VIII contains the following chapters:

- [Appendix A, "Additional Information about Spring and OSGi"](#)
- [Appendix B, "Oracle CEP Schemas"](#)
- [Appendix C, "Schema Reference: EPN Assembly spring-wlevs-v11_1_1_3.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 CEP Metadata Annotation Reference"](#)
- [Appendix J, "Oracle CEP IDE for Eclipse Tutorial"](#)

Additional Information about Spring and OSGi

Oracle Complex Event Processing applications are built on top of the Spring Framework and OSGi Service Platform. Therefore, it is assumed that you are familiar with these technologies and how to program within the frameworks.

For additional information about Spring and OSGi, see:

- Spring Framework API 2.5:
<http://static.springframework.org/spring/docs/2.5.x/api/index.html>
- The Spring Framework - Reference Documentation 2.5:
<http://static.springframework.org/spring/docs/2.5.x/reference/index.html>
- Spring-OSGi Project: <http://www.springframework.org/osgi>
- OSGi Release 4 Service Platform Javadoc:
<http://www.springframework.org/osgi>
- OSGi Release 4 Core Specification: http://www.osgi.org/osgi_technology/download_specs.asp?section=2#Release4



Oracle CEP Schemas

This section contains information on the following subjects:

- [Appendix B.1, "EPN Assembly Schema spring-wlevs-v11_1_1_3.xsd"](#)
- [Appendix B.2, "Component Configuration Schema wlevs_application_config.xsd"](#)
- [Appendix B.3, "Deployment Schema deployment.xsd"](#)
- [Appendix B.4, "Server Configuration Schema wlevs_server_config.xsd"](#)

B.1 EPN Assembly Schema spring-wlevs-v11_1_1_3.xsd

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

The `spring-wlevs-v11_1_1_3.xsd` file describes the structure of EPN assembly files.

This schema file is located in the `ORACLE_CEP_HOME\ocep_11.1\xsd` directory, where `ORACLE_CEP_HOME` is the main Oracle CEP installation directory, such as `d:\oracle_cep`.

For more information, see [Appendix C, "Schema Reference: EPN Assembly spring-wlevs-v11_1_1_3.xsd"](#).

B.1.1 Example EPN Assembly File

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

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns:osgi="http://www.springframework.org/schema/osgi"
  xmlns:wlevs="http://www.bea.com/ns/wlevs/spring"
  xsi:schemaLocation="
    http://www.springframework.org/schema/beans
    http://www.springframework.org/schema/beans/spring-beans.xsd
    http://www.springframework.org/schema/osgi
    http://www.springframework.org/schema/osgi/spring-osgi.xsd
    http://www.bea.com/ns/wlevs/spring
    http://www.bea.com/ns/wlevs/spring/spring-wlevs-v11_1_1_3.xsd">

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

```

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

  <!-- Adapter can be created from a local class, without having to go through a adapter factory -->
  <wlevs:adapter id="helloworldAdapter" class="com.bea.wlevs.adapter.example.helloworld.HelloWorldAdapter" >
    <wlevs:instance-property name="message" value="HelloWorld - the current time is:"/>
  </wlevs:adapter>

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

  <!-- The default processor for Oracle CEP 11.0.0.0 is CQL -->
  <wlevs:processor id="helloworldProcessor" />

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

```

B.2 Component Configuration Schema wlevs_application_config.xsd

An Oracle CEP application contains one or more component configuration files in its META-INF/wlevs directory. You use component configuration files to override the default configuration for Oracle CEP components such as adapters, channels, and processors.

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

- wlevs_base_config.xsd
- wlevs_eventstore_config.xsd
- wlevs_diagnostic_config.xsd

These schema files are located in the *ORACLE_CEP_HOME*\ocep_11.1\xsd directory, where *ORACLE_CEP_HOME* is the main Oracle CEP installation directory, such as d:\oracle_cep.

For more information, see [Appendix D, "Schema Reference: Component Configuration wlevs_application_config.xsd"](#).

B.2.1 Example Component Configuration File

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

```

<?xml version="1.0" encoding="UTF-8"?><n1:config
xmlns:n1="http://www.bea.com/ns/wlevs/config/application"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <processor>
    <name>helloworldProcessor</name>
    <rules>
      <query id="helloworldRule">
        <![CDATA[ select * from helloworldInputChannel [Now] ]]>

```

```

        </query>
    </rules>
</processor>
<channel>
    <name>helloworldInputChannel</name>
    <max-size>10000</max-size>
    <max-threads>2</max-threads>
</channel>
<channel>
    <name>helloworldOutputChannel</name>
    <max-size>10000</max-size>
    <max-threads>2</max-threads>
</channel>
</n1:config>

```

B.3 Deployment Schema deployment.xsd

The deployment file for an Oracle CEP instance is called `deployments.xml` and is located in the `DOMAIN_DIR/servername` directory, where `DOMAIN_DIR` refers to the main domain directory and `servername` refers to the name of the server instance. This XML file lists the OSGi bundles that have been deployed to the server.

The `deployment.xsd` schema file describes the structure of deployment files.

This schema file is located in the `ORACLE_CEP_HOME\ocep_11.1\xsd` directory, where `ORACLE_CEP_HOME` is the main Oracle CEP installation directory, such as `d:\oracle_cep`.

For more information, see [Appendix E, "Schema Reference: Deployment deployment.xsd"](#).

B.3.1 Example Deployment XML File

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

```

<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:wlevs="http://www.bea.com/ns/wlevs/deployment"
    xsi:schemaLocation="
        http://www.springframework.org/schema/beans
        http://www.springframework.org/schema/beans/spring-beans.xsd
        http://www.bea.com/ns/wlevs/deployment
        http://www.bea.com/ns/wlevs/deployment/deployment.xsd">
    <bean class="org.springframework.beans.factory.config.PropertyPlaceholderConfigurer">
        <property name="systemPropertiesModeName" value="SYSTEM_PROPERTIES_MODE_OVERRIDE"/>
    </bean>
    <wlevs:deployment id="fx" state="start"
        location="file:${wlevs.domain.home}/applications/fx/com.bea.wlevs.example.fx_11.1.0.0.jar"/>
</beans>

```

B.4 Server Configuration Schema wlevs_server_config.xsd

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

The `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 CEP installation directory, such as `d:\oracle_cep`.

For more information, see [Appendix F, "Schema Reference: Server Configuration wlevs_server_config.xsd"](#).

B.4.1 Example Server Configuration XML File

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

```
<?xml 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_3.xsd

This appendix describes the elements of the `spring-wlevs-v11_1_1_3.xsd` schema.

For more information, see:

- [Section C.1, "Overview of the Oracle CEP Application Assembly Elements"](#)
- [Section B.1, "EPN Assembly Schema spring-wlevs-v11_1_1_3.xsd"](#)

C.1 Overview of the Oracle CEP Application Assembly Elements

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

C.1.1 Element Hierarchy

The Oracle CEP application assembly elements are organized into the following hierarchy:

```
beans
  Standard Spring and OSGi elements such as bean, osgi-service, and so on.
  wlevs:event-type-repository
    wlevs:event-type
      wlevs:metadata
      wlevs:property
  wlevs:adapter
    wlevs:listener
    wlevs:instance-property
    wlevs:property
  wlevs:processor
    wlevs:listener
    wlevs:source
    wlevs:function
    wlevs:instance-property
    wlevs:property
    wlevs:cache-source
    wlevs:table-source
  wlevs:channel
    wlevs:listener
    wlevs:source
    wlevs:instance-property
```

```
wlevs:property
wlevs:application-timestamped
  wlevs:expression
wlevs:event-bean
  wlevs:listener
  wlevs:instance-property
  wlevs:property
wlevs:factory
wlevs:cache
  wlevs:caching-system
  wlevs:cache-loader
  wlevs:cache-store
  wlevs:cache-listener
wlevs:caching-system
  wlevs:instance-property
  wlevs:property
wlevs:table
```

C.1.2 Example of an EPN Assembly File That Uses Oracle CEP Elements

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

```
<?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_3.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>
```

C.2 wlevs:adapter

Use this element to declare an adapter component to the Spring application context.

C.2.1 Child Elements

The `wlevs:adapter` application assembly element supports the following child elements:

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

C.2.2 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?
<code>id</code>	Unique identifier for this component. This identifier must correspond to the <code><name></code> element in the XML configuration file for this adapter, 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.

Table C-1 (Cont.) Attributes of the wlevs:adapter Application Assembly Element

Attribute	Description	Data Type	Required?
provider	<p>Specifies the adapter service provider. Typically the value of this attribute is a reference to the OSGi-registered adapter factory service.</p> <p>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:</p> <pre>provider="csvgen"</pre> <p>If you are using one of the built-in HTTP publish-subscribe adapters, then specify the following hard-coded values:</p> <ul style="list-style-type: none"> For the built-in pub-sub adapter used for <i>publishing</i>, specify the hard-coded <code>httppub</code> value, such as: <pre>provider="httppub"</pre> For the built-in pub-sub adapter used for <i>subscribing</i>, specify the hard-coded <code>httpsub</code> value, such as: <pre>provider="httpsub"</pre> <p>If you are using a JMS adapter, then specify one of the following hard-coded values:</p> <ul style="list-style-type: none"> For the inbound JMS adapter, specify the <code>jms-inbound</code> value, such as: <pre>provider="jms-inbound"</pre> For the outbound JMS adapter, specify the <code>jms-outbound</code> value, such as: <pre>provider="jms-outbound"</pre> <p>You must specify either the <code>provider</code> or <code>class</code> attribute, but not both, otherwise an exception is raised.</p>	String	No.
class	<p>Specifies the Java class that implements this adapter.</p> <p>You must specify either the <code>provider</code> or <code>class</code> attribute, but not both, otherwise an exception is raised.</p>	String	No
onevent-method	<p>Specifies the method of the adapter implementation that corresponds to the lifecycle <code>onEvent</code> method.</p> <p>Oracle CEP invokes this method when the adapter receives an event.</p>	String	No
init-method	<p>Specifies the method of the adapter implementation that corresponds to the lifecycle <code>init</code> method.</p> <p>Oracle CEP invokes this method after it has set all the supplied instance properties. This method allows the adapter instance to perform initialization only possible when all bean properties have been set and to throw an exception in the event of misconfiguration.</p>	String	No
activate-method	<p>Specifies the method of the adapter implementation that corresponds to the lifecycle <code>activate</code> method.</p> <p>Oracle CEP invokes this method after the dynamic configuration of the adapter has completed. This method allows the adapter instance to perform initialization only possible when all dynamic bean properties have been set and the EPN has been wired.</p>	String	No
suspend-method	<p>Specifies the method of the adapter implementation that corresponds to the lifecycle <code>suspend</code> method.</p> <p>Oracle CEP invokes this method when the application is suspended.</p>	String	No

Table C-1 (Cont.) Attributes of the wlevs:adapter Application Assembly Element

Attribute	Description	Data Type	Required?
destroy-method	Specifies the method of the adapter implementation that corresponds to the lifecycle <code>destroy</code> method. Oracle CEP invokes this method when the application is stopped.	String	No

C.2.3 Example

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

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

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

C.3 wlevs:application-timestamped

Use this element to specify if an `wlevs:channel` is application timestamped, that is, if the application is responsible for assigning a timestamp to each event, using any time domain.

Otherwise, `wlevs:channel` is system timestamped, that is, the Oracle CEP server is responsible for assigning a timestamp to each event using `System.nanoTime`.

C.3.1 Child Elements

The `wlevs:application-timestamped` application assembly element supports the following child elements.

- `wlevs:expression`—Specifies an expression to be used as an application timestamp for event processing.

C.3.2 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 <code>true</code> or <code>false</code> . Default: <code>false</code> . For more information, see "Time" in the <i>Oracle Complex Event Processing CQL Language Reference</i> .	Boolean	No.

C.3.3 Example

The following example shows how to use the `wlevs:application-timestamped` element in the EPN assembly file to specify an implicitly application timestamped channel:

```
<wlevs:channel id="fxMarketAmerOut" >
  <wlevs:application-timestamped>
  </wlevs:application-timestamped>
</wlevs:channel>
```

In the example, the application handles event timestamps internally.

The following example shows how to use `wlevs:application-timestamped` element in the EPN assembly file to specify an explicitly application timestamped channel by specifying the `wlevs:expression` element.

```
<wlevs:channel id="fxMarketAmerOut" >
  <wlevs:application-timestamped>
    <wlevs:expression>mytime+10</wlevs:expression>
  </wlevs:application-timestamped>
</wlevs:channel>
```

In the example, the `wlevs:expression` element defines the arithmetic expression used to assign a timestamp to each event.

C.4 wlevs:cache

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

C.4.1 Child Elements

The `wlevs:cache` application assembly element supports the following child elements.

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

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

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

C.4.2 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?
id	Unique identifier for this component. This identifier must correspond to the <name> element in the XML configuration file for this cache.	String	Yes.
name	Specifies an alternate name for this cache. If not specified, then the name of the cache is the same as its id attribute.	String	No.
key-properties	Specifies a comma-separated list of names of the properties that together form the unique key value for the objects in the cache, or <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 CEP 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 CEP uses the event object itself as both the key and value when it inserts the event object into the cache.	String	No.
key-class	Specifies the name of the Java class used for the cache key when the key is a composite key. If you do not specify the <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 CEP uses the event object itself as both the key and value when it inserts the event object into the cache	String	No.
value-type	Specifies the type for the values contained in the cache. Must be a valid type name in the event type repository. This attribute is required only if the cache is referenced in an Oracle CQL or EPL query. This is because the query processor needs to know the type of events in the cache.	String	No.
caching-system	Specifies the caching system in which this cache is contained. The value of this attribute corresponds to the id attribute of the appropriate <code>wlevs:caching-system</code> element.	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.

C.4.3 Example

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

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

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

C.5 wlevs:cache-listener

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

This element is always a child of `wlevs:cache`.

C.5.1 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.

C.5.2 Example

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

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

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

C.6 wlevs:cache-loader

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

This element is always a child of `wlevs:cache`.

C.6.1 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.

C.6.2 Example

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


```

<wlevs:cache id="cache-id" name="alternative-cache-name">
  <wlevs:caching-system ref="caching-system-id"/>
  <wlevs:cache-loader ref="cache-loader-id" />
</wlevs:cache>
...
<bean id="cache-loader-id" class="wlevs.example.MyCacheLoader"/>

```

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

C.7 wlevs:cache-source

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

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

This element is a child of only `wlevs:processor` element.

C.7.1 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.

C.7.2 Example

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

```

<wlevs:caching-system id="caching-system-id"/>
...
<wlevs:cache id="cache-id"
  name="alternative-cache-name"
  value-type="Company">
  <wlevs:caching-system ref="caching-system-id"/>
</wlevs:cache>
<wlevs:channel id="stream-id"/>
<wlevs:processor id="processor-id">
  <wlevs:cache-source ref="cache-id">
  <wlevs:source ref="stream-id">
</wlevs:processor>

```

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

C.8 wlevs:cache-store

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

This element is always a child of `wlevs:cache`.

C.8.1 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.

C.8.2 Example

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

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

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

C.9 wlevs:caching-system

Specifies the caching system used by the application.

C.9.1 Child Elements

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

- `wlevs:instance-property`
- `wlevs:property`

C.9.2 Attributes

Table C-8 lists the attributes of the `wlevs:caching-system` application assembly element.

Table C-8 Attributes of the wlevs:channel 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><wlevs:channel id="myChannel" provider="coherence" /></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 CEP 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 CEP native caching implementation. If you specify this attribute, it is assumed that the third-party implementation code resides inside the Oracle CEP application bundle itself. The class file to which this attribute points must implement the <code>com.bea.wlevs.cache.api.CachingSystem</code> interface. If you do not specify the provider or class attribute, then the default value is the Oracle CEP native caching implementation for local single-JVM caches; this implementation uses an in-memory store.	String	No

C.9.3 Example

The following example shows the simplest use of the `wlevs:channel` element in the EPN assembly file:

```
<wlevs:channel id="channel-id"/>
```

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

```
<wlevs:channel id="channel-id" provider="channel-provider"/>
<factory id="factory-id" provider-name="channel-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:channel` in turn references it as the caching system implementation for the application.

C.10 wlevs:channel

Use this element to declare a channel to the Spring application context.

By default, channels assume that events are system timestamped. To configure application timestamped events, see child element [wlevs:application-timestamped](#).

C.10.1 Child Elements

The `wlevs:channel` application assembly element supports the following child elements:

- `wlevs:listener`
- `wlevs:source`
- `wlevs:instance-property`
- `wlevs:property`
- `wlevs:application-timestamped`

C.10.2 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 Section 9.1.6, "Batch Processing Channels" .	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><name></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 > 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 > 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 Section 9.1.2.2, "Channels as Relations".</p>	String	No.
provider	<p>Specifies the streaming provider.</p> <p>Valid values are:</p> <ul style="list-style-type: none"> ▪ <code>oracle.channel</code> <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.

C.10.3 Example

The following example shows how to use the `wlevs:channel` element in the EPN assembly file:

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

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

C.11 wlevs:event-bean

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

For example, you can monitor an event bean using the Oracle CEP monitoring framework, make use of the Configuration framework metadata annotations, and record and playback events that pass through the event bean. An event-bean can also

participate in the Oracle CEP bean lifecycle by specifying methods in its EPN assembly file declaration, rather than by implementing Oracle CEP API interfaces.

C.11.1 Child Elements

The `wlevs:event-bean` application assembly element supports the following child elements:

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

C.11.2 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><name></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 CEP 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.
<code>onevent-method</code>	Specifies the method of the event bean implementation that corresponds to the lifecycle <code>onEvent</code> method. Oracle CEP invokes this method when the event bean receives an event. By using this lifecycle attribute, the event bean implementation does not have to explicitly implement an Oracle CEP interface.	String	No
<code>init-method</code>	Specifies the method of the event bean implementation that corresponds to the lifecycle <code>init</code> method. Oracle CEP invokes this method after it has set all the supplied instance properties. This method allows the bean instance to perform initialization only possible when all bean properties have been set and to throw an exception in the event of misconfiguration. By using this lifecycle attribute, the event bean implementation does not have to explicitly implement an Oracle CEP interface.	String	No

Table C–10 (Cont.) Attributes of the wlevs:event-bean Application Assembly Element

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

C.11.3 Example

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

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

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

C.12 wlevs:event-type-repository

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

This element does not have any attributes.

C.12.1 Child Elements

The `wlevs:event-type-repository` application assembly element supports the following child element:

- `wlevs:event-type`

C.12.2 Example

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

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

```

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

```

In the example, the `wlevs:event-type-repository` element groups a single `wlevs:event-type` element to declare a single event type: `HelloWorldEvent`. See [Section C.13, "wlevs:event-type"](#) for additional details.

C.13 wlevs:event-type

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

You can define an event type in the following ways:

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

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

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

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

C.13.1 Child Elements

The `wlevs:event-type` application assembly element supports the following child elements:

- `wlevs:metadata`
- `wlevs:property`

C.13.2 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?
<code>id</code>	Specifies the unique identifier for this event type. If you do not specify this attribute, Oracle CEP automatically generates an identifier for you.	String	No.
<code>type-name</code>	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.

C.13.3 Example

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

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

In the example, the name of the event type is `SimpleEvent` and its definition is determined by the `wlevs:property` elements. The values for the type attribute must conform to the `com.bea.wlevs.ede.api.Type` class.

For more information, see [Section 2.1.3, "Event Type Data Types"](#).

C.14 wlevs:expression

Use this element to specify an arithmetic expression in [wlevs:application-timestamped](#) to be used as an application timestamp for event processing.

For more information, see "arith_expr" in the *Oracle Complex Event Processing CQL Language Reference*.

C.14.1 Example

The following example shows how to use `wlevs:expression` element in the EPN assembly file to specify an explicitly application timestamped channel.

```
<wlevs:channel id="fxMarketAmerOut" >
  <wlevs:application-timestamped>
    <wlevs:expression>mytime + 10</wlevs:expression>
  </wlevs:application-timestamped>
</wlevs:channel>
```

In the example, the `wlevs:expression` element defines the arithmetic expression used to assign a timestamp to each event.

C.15 wlevs:factory

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

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

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

This element does not have any child elements.

C.15.1 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.

C.15.2 Example

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

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

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

C.16 `wlevs:function`

Use this element to specify a bean that contains user-defined functions for a processor. Oracle CEP supports both single-row and aggregate functions.

This element always has a standard Spring bean element either as a child or as a reference that specifies the Spring bean that implements the user-defined function.

For a single-row function for an Oracle CQL processor, you may specify one method on the implementing class as the function using the `exec-method` attribute. In this case, the method must be public and must be uniquely identifiable by its name (that is, the method cannot have been overridden). You may define an alias for the `exec-method` name using the `function-name` attribute. In the Oracle CQL query, you may call only the `exec-method` (either by its name or the `function-name` alias).

For a single-row function on an EPL processor, you may define an alias for the implementing class name using the `function-name` attribute. The `exec-method` name is not applicable in this case. In the EPL query, you may call any public or static method on the implementing class using either the implementing class name or the `function-name` alias.

For an aggregate function on either an Oracle CQL or EPL processor, the Spring bean must implement the following interfaces from the `com.bea.wlevs.processor` package:

- `AggregationFunctionFactory`
- `AggregationFunction`

For an aggregate function, the `exec-method` attribute is not applicable on both an Oracle CQL processor and an EPL processor.

For more information, see:

- "User-Defined Functions" in the *Oracle Complex Event Processing CQL Language Reference*

- "EPL Reference: Functions" in the *Oracle Complex Event Processing EPL Language Reference*

C.16.1 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 <code>Spring bean</code> 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 <code>Spring bean</code> class name. You can then use the <code>function-name</code> in your EPL query instead of the <code>Spring bean</code> class name and still invoke any public or static method that the <code>Spring bean</code> 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 <code>Spring bean</code> class name. You can then use the <code>function-name</code> in your EPL query instead of the <code>Spring bean</code> class name. The default value is the <code>Spring bean</code> name.	String	No.
<code>ref</code>	Specifies the <code>Spring bean</code> that implements the function. Set this attribute to the value of the <code>id</code> attribute of the <code>Spring bean</code> . This is an alternative to making the <code>Spring bean</code> element a child of the <code>wlevs:function</code> element.	String	No.

C.16.2 Example

The following examples show how to use the `wlevs:function` element and its attributes on both Oracle CQL and EPL processors:

- [Section C.16.2.1, "Single-Row User-Defined Function on an Oracle CQL Processor"](#)
- [Section C.16.2.2, "Single-Row User-Defined Function on an EPL Processor"](#)
- [Section C.16.2.3, "Aggregate User-Defined Function on an Oracle CQL Processor"](#)
- [Section C.16.2.4, "Aggregate User-Defined Function on an EPL Processor"](#)
- [Section C.16.2.5, "Specifying the Implementation Class: Nested Bean or Reference"](#)

C.16.2.1 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>
...
```

C.16.2.2 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>

```

C.16.2.3 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>
...

```

C.16.2.4 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>
...

```

C.16.2.5 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"/>

```


C.17 wlevs:instance-property

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

For example, when you specify an `wlevs:instance-property` for a `wlevs:event-bean`, Oracle CEP 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>.

C.17.1 Child Elements

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

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

C.17.2 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?
<code>name</code>	Specifies the name of the property, following JavaBean naming conventions.	String	Yes.
<code>ref</code>	A short-cut alternative to a nested <code><ref bean='...' /></code> element.	String	No.
<code>value</code>	A short-cut alternative to a nested <code><value>...</value></code> element.	String	No.

C.17.3 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 CEP will throw an `Invalid Property` exception. In this case, the setter name should be `setObj`.

C.18 wlevs:listener

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

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

This element is always a child of `wlevs:adapter`, `wlevs:processor`, `wlevs:channel`, or `wlevs:caching-system`.

C.18.1 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?
<code>ref</code>	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.	<code>String</code>	No.

C.18.2 Example

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

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

In the example, the `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`.

C.19 wlevs:metadata

Specifies the definition of an event type by listing its fields as a group of Spring entry elements. When you define an event type this way, Oracle CEP automatically generates the Java class for you.

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

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

The `wlevs:metadata` element is defined as the Spring `mapType` type; for additional details of this Spring data type, see the <http://www.springframework.org/schema/beans/spring-beans-2.0.xsd>.

C.19.1 Child Elements

The `wlevs:metadata` element can have one or more standard Spring entry child elements as defined in the <http://www.springframework.org/schema/beans/spring-beans-2.0.xsd>.

C.19.2 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?
<code>key-type</code>	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 <code>entry</code> elements.	String	No.

C.19.3 Example

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

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

```

...
</wlevs:event-type>

```

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

C.20 wlevs:processor

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

C.20.1 Child Elements

The `wlevs:processor` Spring element supports the following child elements:

- `wlevs:instance-property`
- `wlevs:listener`
- `wlevs:property`
- `wlevs:cache-source`
- `wlevs:table-source`
- `wlevs:function`

C.20.2 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?
<code>id</code>	Unique identifier for this component. This identifier must correspond to the <code><name></code> element in the XML configuration file for this processor; this is how Oracle CEP knows which Oracle CQL or EPL rules to execute for which processor component in your network.	String	Yes.
<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>provider</code>	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"> ■ <code>epl</code> ■ <code>cql</code> The default value is <code>cql</code> .	String	No.
<code>queryURL</code>	Specifies a URL that points to an Oracle CQL or EPL rules definition file for this processor.	String.	No.

C.20.3 Example

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

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

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

C.21 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 CEP 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>.

C.21.1 Child Elements

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

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

C.21.2 Attributes

[Table C-18](#) lists the attributes of the `wlevs:property` application assembly element.

Table C–18 Attributes of the wlevs:property Application Assembly Element

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

C.21.3 Example

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

```
<wlevs:event-type type-name="ForeignExchangeEvent">
  <wlevs:metadata>
    <entry key="symbol" value="java.lang.String"/>
    <entry key="price" value="java.lang.Double"/>
  </wlevs:metadata>
  <wlevs:property name="builderFactory">
    <bean id="builderFactory"
      class="com.bea.wlevs.example.fx.ForeignExchangeBuilderFactory"/>
  </wlevs:property>
</wlevs:event-type>
```

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

C.22 wlevs:source

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

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

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

This element is a child of `wlevs:channel` or `wlevs:processor`.

C.22.1 Attributes

[Table C–19](#) lists the attributes of the `wlevs:source` application assembly element.

Table C–19 Attributes of the wlevs:source Application Assembly Element

Attribute	Description	Data Type	Required?
ref	Specifies the source of the channel to which this element is a child. Set this attribute to the value of the <code>id</code> attribute of the source component. You do not specify this attribute if you are nesting sources.	String	No.

C.22.2 Example

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

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

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

C.23 wlevs:table

Specifies a relational database table that supplies data to one or more processor components. The processor components in turn are associated with an Oracle CQL query that directly references the table.

C.23.1 Attributes

Table C-20 lists the attributes of the `wlevs:table` application assembly element.

Table C-20 Attributes of the `wlevs:table` Application Assembly Element

Attribute	Description	Data Type	Required?
<code>id</code>	Unique identifier for this component. This identifier must correspond to the <code><name></code> element in the XML configuration file for this table.	String	Yes.
<code>event-type</code>	The name of the event type associated with this table as defined in the event type repository.	String	Yes.
<code>data-source</code>	The name of the relational data source defined in the Oracle CEP server configuration file used to access this database table.	String	Yes.

C.23.2 Example

The following example shows how to use the `wlevs:table` element in the EPN assembly file:

```
<wlevs:table id="Stock" event-type="StockEvent" data-source="StockDs" />

<wlevs:processor id="proc">
  <wlevs:table-source ref="Stock" />
</wlevs:processor>
```

In this example, a `wlevs:processor` references the table using its `wlevs:table-source` element.

C.24 wlevs:table-source

Specifies a relational database table that supplies data to this processor component. The processor component in turn is associated with an Oracle CQL query that directly references the table.

This element is a child of only `wlevs:processor` element.

C.24.1 Attributes

Table C–21 lists the attributes of the `wlevs:table-source` application assembly element.

Table C–21 Attributes of the `wlevs:table-source` Application Assembly Element

Attribute	Description	Data Type	Required?
<code>ref</code>	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.

C.24.2 Example

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

```
<wlevs:table id="Stock" event-type="StockEvent" data-source="StockDs" />

<wlevs:processor id="proc">
  <wlevs:table-source ref="Stock" />
</wlevs:processor>
```

Schema Reference: Component Configuration `wlevs_application_config.xsd`

This appendix describes the elements of the `wlevs_application_config.xsd` schema.

For more information, see:

- [Section D.1, "Overview of the Oracle CEP Component Configuration Elements"](#)
- [Section B.2, "Component Configuration Schema `wlevs_application_config.xsd`"](#).

D.1 Overview of the Oracle CEP Component Configuration Elements

Oracle CEP provides a number of component configuration elements that you use to define the characteristics of the of the components you declare in the EPN assembly file.

D.1.1 Element Hierarchy

The top-level Oracle CEP 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

```

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
    max-size
    max-threads
    selector
    heartbeat

```

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

```

D.1.2 Example of an Oracle CEP Component Configuration File

The following sample component configuration file from the HelloWorld application shows how to use many of the Oracle CEP elements:

```

<?xml 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>

```

D.2 accept-backlog

Use this element to define the maximum number of pending connections allowed on a socket. This element is only applicable in a [netio](#) element.

D.2.1 Child Elements

The `accept-backlog` component configuration element has no child elements.

D.2.2 Attributes

The `accept-backlog` component has no attributes.

D.2.3 Example

The following example shows how to use the `accept-backlog` element in the component configuration file:

```

<netio>
    <provider-name>providerCache</provider-name>
    <num-threads>1000</num-threads>
    <accept-backlog>50</accept-backlog>
</netio>

```

D.3 adapter

Use this element to define a custom adapter component. For an HTTP publish-subscribe or JMS adapter, use the specific [http-pub-sub-adapter](#) and [jms-adapter](#) elements.

For more information, see [Chapter 14, "Configuring Custom Adapters"](#).

D.3.1 Child Elements

The `adapter` component configuration element supports the following child elements:

- [name](#)
- [record-parameters](#)
- [playback-parameters](#)
- [symbols](#)
- [work-manager-name](#)
- [netio](#)

D.3.2 Attributes

The `adapter` component configuration element has no attributes.

D.3.3 Example

The following example shows how to use the `adapter` element in the component configuration file:

```
<adapter>
  <name>trackdata</name>
  <symbols>
    <symbol>BEAS</symbol>
    <symbol>IBM</symbol>
  </symbols>
</adapter>
```

In the example, the adapter's unique identifier is `trackdata`.

D.4 amount

Use this element to define the a time duration of a diagnostic profile. This element is applicable in any of the following elements:

- [average-interval](#)
- [collect-interval](#)
- [throughput-interval](#)

For more information, see "Monitoring the Throughput and Latency of a Stage or Path in the EPN" in the *Oracle Complex Event Processing Visualizer User's Guide*.

D.4.1 Child Elements

The `amount` component configuration has no child elements:

D.4.2 Attributes

The `amount` component has no attributes.

D.4.3 Example

The following example shows how to use the `amount` element in the component configuration file:

```
<diagnostic-profiles>
  <name>myselfprofiles</name>
  <profile>
    <name>testProfile0</name>
    <enabled>true</enabled>
    <start-stage>MetricSubscriber</start-stage>
    <max-latency>
      <collect-interval>
        <amount>1000</amount>
        <unit>s</unit>
      </collect-interval>
    <name>testProfile0MaxLat</name>
    <start-location>
      <application>diagnostic</application>
      <stage>MetricSubscriber</stage>
      <direction>INBOUND</direction>
```

```

        </start-location>
        <end-location>
            <application>diagnostic</application>
            <stage>MonitorProcessor</stage>
            <direction>OUTBOUND</direction>
        </end-location>
    </max-latency>
</profile>
</diagnostic-profiles>

```

D.5 application

Use this element to define the type of application Oracle CEP server applies to a foreign stage. In a diagnostic profile, this element always has a value of `diagnostic`.

For more information, see "Monitoring the Throughput and Latency of a Stage or Path in the EPN" in the *Oracle Complex Event Processing Visualizer User's Guide*.

D.5.1 Child Elements

The `application` component configuration has no child elements:

D.5.2 Attributes

The `application` component has no attributes.

D.5.3 Example

The following example shows how to use the `application` element in the component configuration file:

```

<diagnostic-profiles>
  <name>myselfprofiles</name>
  <profile>
    <name>testProfile0</name>
    <enabled>true</enabled>
    <start-stage>MetricSubscriber</start-stage>
    <max-latency>
      <collect-interval>
        <amount>1000</amount>
        <unit>s</unit>
      </collect-interval>
      <name>testProfile0MaxLat</name>
      <start-location>
        <application>diagnostic</application>
        <stage>MetricSubscriber</stage>
        <direction>INBOUND</direction>
      </start-location>
      <end-location>
        <application>diagnostic</application>
        <stage>MonitorProcessor</stage>
        <direction>OUTBOUND</direction>
      </end-location>
    </max-latency>
  </profile>
</diagnostic-profiles>

```

D.6 average-interval

Use this element to define the time interval for which you want to gather metrics.

For more information, see "Monitoring the Throughput and Latency of a Stage or Path in the EPN" in the *Oracle Complex Event Processing Visualizer User's Guide*.

D.6.1 Child Elements

The `average-interval` component configuration element supports the following child elements:

- `amount`
- `unit`

D.6.2 Attributes

The `average-interval` component has no attributes.

D.6.3 Example

The following example shows how to use the `average-interval` element in the component configuration file:

```
<diagnostic-profiles>
  <name>myselfprofiles</name>
  <profile>
    <name>testProfile0</name>
    <enabled>true</enabled>
    <start-stage>MetricSubscriber</start-stage>
    <throughput>
      <throughput-interval>
        <amount>100000</amount>
        <unit>MICROSECONDS</unit>
      </throughput-interval>
      <average-interval>
        <amount>100000000</amount>
        <unit>NANOSECONDS</unit>
      </average-interval>
      <location>
        <application>diagnostic</application>
        <stage>AlertEventStream</stage>
        <direction>INBOUND</direction>
      </location>
    </throughput>
  </profile>
</diagnostic-profiles>
```

D.7 average-latency

Use this element to define an average latency calculation in a diagnostic profile.

For more information, see "Monitoring the Throughput and Latency of a Stage or Path in the EPN" in the *Oracle Complex Event Processing Visualizer User's Guide*.

D.7.1 Child Elements

The `average-latency` component configuration element supports the following child elements:

- `name`
- `collect-interval`
- `start-location`

- `end-location`
- `threshold`

D.7.2 Attributes

The `average-latency` component has no attributes.

D.7.3 Example

The following example shows how to use the `average-latency` element in the component configuration file:

```
<diagnostic-profiles>
  <name>myselfprofiles</name>
  <profile>
    <name>testProfile0</name>
    <enabled>true</enabled>
    <start-stage>MetricSubscriber</start-stage>
    <average-latency>
      <start-location>
        <application>diagnostic</application>
        <stage>MetricSubscriber</stage>
        <direction>INBOUND</direction>
      </start-location>
      <end-location>
        <application>diagnostic</application>
        <stage>MonitorProcessor</stage>
        <direction>OUTBOUND</direction>
      </end-location>
      <threshold>
        <amount>100</amount>
        <unit>MILLISECONDS</unit>
      </threshold>
    </average-latency>
  </profile>
</diagnostic-profiles>
```

D.8 batch-size

Use this element to define the number of updates that are picked up from the store buffer to write back to the backing store. This element may be changed dynamically.

D.8.1 Child Elements

The `batch-size` component configuration element has no child elements.

D.8.2 Attributes

The `batch-size` component has no attributes.

D.8.3 Example

The following example shows how to use the `batch-size` element in the component configuration file:

```
<record-parameters>
  <dataset-name>tuple1</dataset-name>
  <event-type-list>
    <event-type>TupleEvent1</event-type>
  </event-type-list>
  <provider-name>test-rdbms-provider</provider-name>
```

```
<store-policy-parameters>
  <parameter>
    <name>timeout</name>
    <value>300</value>
  </parameter>
</store-policy-parameters>
<batch-size>1</batch-size>
<batch-time-out>10</batch-time-out>
</record-parameters>
```

D.9 batch-time-out

Use this element to define The number of seconds event buffer will wait to accumulate `batch-size` number of events before to write to the event store.

D.9.1 Child Elements

The `batch-time-out` component configuration element has no child elements.

D.9.2 Attributes

The `batch-time-out` component has no attributes.

D.9.3 Example

The following example shows how to use the `batch-time-out` element in the component configuration file:

```
<record-parameters>
  <dataset-name>tuple1</dataset-name>
  <event-type-list>
    <event-type>TupleEvent1</event-type>
  </event-type-list>
  <provider-name>test-rdbms-provider</provider-name>
  <store-policy-parameters>
    <parameter>
      <name>timeout</name>
      <value>300</value>
    </parameter>
  </store-policy-parameters>
  <batch-size>1</batch-size>
  <batch-time-out>10</batch-time-out>
</record-parameters>
```

D.10 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 Complex Event Processing CQL Language Reference*
- "Parameterized Queries" in the *Oracle Complex Event Processing EPL Language Reference*

D.10.1 Child Elements

The `binding` component configuration element supports the following child elements:

- [params](#)

D.10.2 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.

D.10.3 Example

The following example shows how to use the binding element in the component configuration file:

```
<processor>
  <name>processor1</name>
  <record-parameters>
    <dataset-name>test1data</dataset-name>
    <event-type-list>
      <event-type>SimpleEvent</event-type>
    </event-type-list>
    <provider-name>test-rdbms-provider</provider-name>
    <batch-size>1</batch-size>
    <batch-time-out>10</batch-time-out>
  </record-parameters>
  <rules>
    <rule id="rule1"><![CDATA[
      select stockSymbol, avg(price) as percentage
      from StockTick retain 5 events
      where stockSymbol=?
      having avg(price) > ? or avg(price) < ?
    ]]></rule>
  </rules>
  <bindings>
    <binding id="rule1">
      <params>BEAS,10.0,-10.0</params>
      <params id="IBM">IBM,5.0,5.0</params>
    </binding>
  </bindings>
</processor>
```

D.11 bindings (jms-adapter)

Using the `com.oracle.cep.cluster.hagroups.ActiveActiveGroupBean`, you can partition an incoming JMS stream in Oracle CEP 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 22.2.2, "ActiveActiveGroupBean"](#).

D.11.1 Child Elements

The `bindings` component configuration element supports the following child elements:

- [group-binding](#)

D.11.2 Attributes

The bindings component has no attributes.

D.11.3 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 CEP 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.

D.12 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 Complex Event Processing CQL Language Reference*
- "Parameterized Queries" in the *Oracle Complex Event Processing EPL Language Reference*

D.12.1 Child Elements

The bindings component configuration element supports the following child elements:

- [binding](#)

D.12.2 Attributes

The `bindings` component has no attributes.

D.12.3 Example

The following example shows how to use the `bindings` element in the component configuration file:

```
<processor>
  <name>processor1</name>
  <record-parameters>
    <dataset-name>test1data</dataset-name>
    <event-type-list>
      <event-type>SimpleEvent</event-type>
    </event-type-list>
    <provider-name>test-rdbms-provider</provider-name>
    <batch-size>1</batch-size>
    <batch-time-out>10</batch-time-out>
  </record-parameters>
  <rules>
    <rule id="rule1"><![CDATA[
      select stockSymbol, avg(price) as percentage
      from StockTick retain 5 events
      where stockSymbol=?
      having avg(price) > ? or avg(price) < ?
    ]]></rule>
  </rules>
  <bindings>
    <binding id="rule1">
      <params>BEAS,10.0,-10.0</params>
      <params id="IBM">IBM,5.0,5.0</params>
    </binding>
  </bindings>
</processor>
```

D.13 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.

D.13.1 Child Elements

The `buffer-size` component configuration element has no child elements.

D.13.2 Attributes

The `buffer-size` component has no attributes.

D.13.3 Example

The following example shows how to use the `buffer-size` element in the component configuration file:

```
<caching-system>
  <name>providerCachingSystem</name>
  <cache>
    <name>providerCache</name>
    <max-size>1000</max-size>
    <eviction-policy>FIFO</eviction-policy>
    <time-to-live>60000</time-to-live>
  </cache>
</caching-system>
```

```
<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>
```

D.14 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.

D.14.1 Child Elements

The `buffer-write-attempts` component configuration element has no child elements.

D.14.2 Attributes

The `buffer-write-attempts` component has no attributes.

D.14.3 Example

The following example shows how to use the `buffer-write-attempts` element in the component configuration file:

```
<caching-system>
  <name>providerCachingSystem</name>
  <cache>
    <name>providerCache</name>
    <max-size>1000</max-size>
    <eviction-policy>FIFO</eviction-policy>
    <time-to-live>60000</time-to-live>
    <idle-time>120000</idle-time>
    <write-behind>
      <work-manager-name>JettyWorkManager</work-manager-name>
      <batch-size>100</batch-size>
      <buffer-size>100</buffer-size>
      <buffer-write-attempts>100</buffer-write-attempts>
      <buffer-write-timeout>100</buffer-write-timeout>
    </write-behind>
    <work-manager-name>JettyWorkManager</work-manager-name>
    <listeners asynchronous="false">
      <work-manager-name>JettyWorkManager</work-manager-name>
    </listeners>
  </cache>
</caching-system>
```

D.15 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.

D.15.1 Child Elements

The `buffer-write-timeout` component configuration element has no child elements.

D.15.2 Attributes

The `buffer-write-timeout` component has no attributes.

D.15.3 Example

The following example shows how to use the `buffer-write-timeout` element in the component configuration file:

```
<caching-system>
  <name>providerCachingSystem</name>
  <cache>
    <name>providerCache</name>
    <max-size>1000</max-size>
    <eviction-policy>FIFO</eviction-policy>
    <time-to-live>60000</time-to-live>
    <idle-time>120000</idle-time>
    <write-behind>
      <work-manager-name>JettyWorkManager</work-manager-name>
      <batch-size>100</batch-size>
      <buffer-size>100</buffer-size>
      <buffer-write-attempts>100</buffer-write-attempts>
      <buffer-write-timeout>100</buffer-write-timeout>
    </write-behind>
    <work-manager-name>JettyWorkManager</work-manager-name>
    <listeners asynchronous="false">
      <work-manager-name>JettyWorkManager</work-manager-name>
    </listeners>
  </cache>
</caching-system>
```

D.16 cache

Use this element to define a cache for a component. A *cache* is a temporary storage area for events, created exclusively to improve the overall performance of your Oracle CEP application; it is not necessary for the application to function correctly. Oracle CEP applications can optionally publish or consume events to and from a cache to increase the availability of the events and increase the performance of their applications.

For more information, see [Section 12.2, "Configuring an Oracle CEP Local Caching System and Cache"](#).

D.16.1 Child Elements

The `cache` component configuration element supports the following child elements:

- [name](#)

- `max-size`
- `eviction-policy`
- `time-to-live`
- `idle-time`
- One of:
 - `write-none`
 - `write-through`
 - `write-behind`
- `work-manager-name`
- `listeners`

D.16.2 Attributes

The cache component has no attributes.

D.16.3 Example

The following example shows how to use the `cache` element in the component configuration file:

```
< caching-system >
  < name > providerCachingSystem < /name >
  < cache >
    < name > providerCache < /name >
    < max-size > 1000 < /max-size >
    < eviction-policy > FIFO < /eviction-policy >
    < time-to-live > 60000 < /time-to-live >
    < idle-time > 120000 < /idle-time >
    < write-none / >
    < work-manager-name > JettyWorkManager < /work-manager-name >
    < listeners asynchronous="false" >
      < work-manager-name > JettyWorkManager < /work-manager-name >
    < /listeners >
  < /cache >
< / caching-system >
```

D.17 caching-system

Use this element to define an Oracle CEP local caching system component. A *caching system* refers to a configured instance of a caching implementation. A caching system defines a named set of configured caches as well as the configuration for remote communication if any of the caches are distributed across multiple machines.

For more information, see [Section 12.2.1, "Configuring an Oracle CEP Local Cache as an Event Listener"](#).

D.17.1 Child Elements

The `caching-system` component configuration element supports the following child elements:

- `name`
- `cache`

D.17.2 Attributes

The `channel-caching-system` component has no attributes.

D.17.3 Example

The following example shows how to use the `channel-caching-system` element in the component configuration file:

```
<channel-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>
</channel-caching-system>
```

In the example, the channel's unique identifier is `providerCachingSystem`.

D.18 channel

Use this element to define a channel component. An Oracle CEP application contains one or more channel components that represent the physical conduit through which events flow between other types of components, such as between adapters and processors, and between processors and event beans (business logic POJOs).

D.18.1 Child Elements

The `channel` component configuration element supports the following child elements:

- `name`
- `record-parameters`
- `playback-parameters`
- `max-size`
- `max-threads`
- `selector`
- `heartbeat`

D.18.2 Attributes

The `channel` component has no attributes.

D.18.3 Example

The following example shows how to use the `channel` element in the component configuration file:

```
<channel>
```

```
<name>MatchOutputChannel</name>
<max-size>0</max-size>
<max-threads>0</max-threads>
<selector>match</selector>
</channel>
```

In the example, the channel's unique identifier is `MatchOutputChannel`.

D.19 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.

D.19.1 Child Elements

The `channel` component configuration element has no child elements.

D.19.2 Attributes

The `channel` component has no attributes.

D.19.3 Example

The following example shows how to use the `channel` element in the component configuration file:

```
<http-pub-sub-adapter>
  <name>localPublisher</name>
  <server-context-path>/pubsub</server-context-path>
  <channel>/channel2</channel>
</http-pub-sub-adapter>
```

In the example, the `localPublisher` pub-sub adapter publishes to a local channel with pattern `/channel2`.

D.20 coherence-cache-config

Use this element to define the Oracle Coherence cache configuration for a [coherence-caching-system](#).

For more information, see [Section 12.2, "Configuring an Oracle CEP Local Caching System and Cache"](#).

D.20.1 Child Elements

The `coherence-cache-config` component configuration element has no child elements.

D.20.2 Attributes

The `coherence-cache-config` component has no attributes.

D.20.3 Example

The following example shows how to use the `coherence-cache-config` element in the component configuration file:


```

<coherence-caching-system>
  <name>nativeCachingSystem</name>
  <coherence-cache-config>
    applications/cache_cql/coherence/coherence-cache-config.xml
  </coherence-cache-config></coherence-caching-system>

```

D.21 coherence-caching-system

Use this element to define an Oracle Coherence caching system component. A *caching system* refers to a configured instance of a caching implementation. A caching system defines a named set of configured caches as well as the configuration for remote communication if any of the caches are distributed across multiple machines.

For more information, see [Section 12.3, "Configuring an Oracle Coherence Caching System and Cache"](#).

D.21.1 Child Elements

The `coherence-caching-system` component configuration element supports the following child elements:

- [name](#)
- [coherence-cache-config](#)
- [coherence-cluster-config](#)

D.21.2 Attributes

The `coherence-caching-system` component has no attributes.

D.21.3 Example

The following example shows how to use the `coherence-caching-system` element in the component configuration file:

```

<coherence-caching-system>
  <name>nativeCachingSystem</name>
  <coherence-cache-config>
    applications/cache_cql/coherence/coherence-cache-config.xml
  </coherence-cache-config>
</coherence-caching-system>

```

In the example, the channel's unique identifier is `nativeCachingSystem`.

D.22 coherence-cluster-config

Use this element to define the Oracle Coherence cluster configuration for a [coherence-caching-system](#).

For more information, see "Overview of Oracle CEP Multi-Server Domain Administration" in the *Oracle Complex Event Processing Administrator's Guide*.

D.22.1 Child Elements

The `coherence-cache-config` component configuration element has no child elements.

D.22.2 Attributes

The `coherence-cache-config` component has no attributes.

D.22.3 Example

The following example shows how to use the `coherence-cache-config` element in the component configuration file:

```
<coherence-caching-system>
  <name>nativeCachingSystem</name>
  <coherence-cluster-config>
    applications/cluster_cql/coherence/coherence-cluster-config.xml
  </coherence-cluster-config></coherence-caching-system>
```

D.23 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 Complex Event Processing Visualizer User's Guide*.

D.23.1 Child Elements

The `collect-interval` component configuration element supports the following child elements:

- `amount`
- `unit`

D.23.2 Attributes

The `collect-interval` component has no attributes.

D.23.3 Example

The following example shows how to use the `collect-interval` element in the component configuration file:

```
<diagnostic-profiles>
  <name>myselfprofiles</name>
  <profile>
    <name>testProfile0</name>
    <enabled>true</enabled>
    <start-stage>MetricSubscriber</start-stage>
    <max-latency>
      <collect-interval>
        <amount>1000</amount>
        <unit>s</unit>
      </collect-interval>
    <name>testProfile0MaxLat</name>
    <start-location>
      <application>diagnostic</application>
      <stage>MetricSubscriber</stage>
      <direction>INBOUND</direction>
    </start-location>
    <end-location>
      <application>diagnostic</application>
      <stage>MonitorProcessor</stage>
      <direction>OUTBOUND</direction>
    </end-location>
  </profile>
</diagnostic-profiles>
```

```

        </end-location>
    </max-latency>
</profile>
</diagnostic-profiles>

```

D.24 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 7.3, "Creating a Custom Converter Between JMS Messages and Event Types"](#)
- [Section D.101, "work-manager"](#)

D.24.1 Child Elements

The `concurrent-consumers` component configuration element has no child elements.

D.24.2 Attributes

The `concurrent-consumers` component has no attributes.

D.24.3 Example

The following example shows how to use the `concurrent-consumers` element in the component configuration file:

```

<jms-adapter>
  <name>jmsInbound-text</name>
  <connection-jndi-name>weblogic.jms.ConnectionFactory</connection-jndi-name>
  <destination-name>JMSServer-0/Module1!Queue1</destination-name>
  <user>weblogic</user>
  <password>weblogic</password>
  <work-manager>JettyWorkManager</work-manager>
  <concurrent-consumers>1</concurrent-consumers>
  <session-transacted>>false</session-transacted>
</jms-adapter>

```

D.25 connection-jndi-name

Use this optional element to define a JNDI name of the JMS connection factory. Default value is `weblogic.jms.ConnectionFactory` for Oracle CEP server JMS.

D.25.1 Child Elements

The `connection-jndi-name` component configuration element has no child elements.

D.25.2 Attributes

The `connection-jndi-name` component has no attributes.

D.25.3 Example

The following example shows how to use the `connection-jndi-name` element in the component configuration file:

```
<jms-adapter>
  <name>jmsInbound-text</name>
  <connection-jndi-name>weblogic.jms.ConnectionFactory</connection-jndi-name>
  <destination-name>JMSServer-0/Module1!Queue1</destination-name>
  <user>weblogic</user>
  <password>weblogic</password>
  <work-manager>JettyWorkManager</work-manager>
  <concurrent-consumers>1</concurrent-consumers>
  <session-transacted>>false</session-transacted>
</jms-adapter>
```

D.26 connection-encrypted-password

Use the `connection-encrypted-password` element to define the encrypted [jms-adapter](#) password that Oracle CEP uses when it acquires a connection to the JMS service provider.

When Oracle CEP 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 CEP 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 7.4, "Encrypting Passwords in the JMS Adapter Component Configuration File"](#).

D.26.1 Child Elements

The `connection-encrypted-password` component configuration element has no child elements.

D.26.2 Attributes

The `connection-encrypted-password` component has no attributes.

D.26.3 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>

```

D.27 connection-password

Use the `connection-password` element to define the `jms-adapter` password that Oracle CEP uses when it acquires a connection to the JMS service provider.

When Oracle CEP 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 CEP uses the `user` and `password` (or `encrypted-password`) elements.

Use either `connection-password` or `connection-encrypted-password` but not both.

D.27.1 Child Elements

The `connection-password` component configuration element has no child elements.

D.27.2 Attributes

The `connection-password` component has no attributes.

D.27.3 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>

```

D.28 connection-user

Use the `connection-user` element to define the `jms-adapter` user name that Oracle CEP uses when it acquires a connection to the JMS service provider.

When Oracle CEP 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 CEP 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.

D.28.1 Child Elements

The `connection-user` component configuration element has no child elements.

D.28.2 Attributes

The `connection-user` component has no attributes.

D.28.3 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>
```

D.29 database

Use this element to define a database reference for an EPL processor component.

For more information, see [Chapter 11, "Configuring EPL Processors"](#).

D.29.1 Child Elements

The database component configuration element has no child elements.

D.29.2 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?
<code>name</code>	Unique identifier for this query.	String	Yes.
<code>data-source-name</code>	The name of the data source as defined in the Oracle CEP server config.xml file.	String	Yes.

D.29.3 Example

The following example shows how to use the database element in the component configuration file:

```
<processor>
  <name>proc</name>
  <rules>
    <rule id="rule1"><![CDATA[
      SELECT symbol, price
      FROM ExchangeEvent retain 1 event,
      StockDb ('SELECT symbol FROM Stock WHERE symbol = ${symbol}')]>
```

```

    ]]></rule>
</rules>

<database name="StockDb" data-source-name="StockDs" />

</processor>

```

D.30 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.

D.30.1 Child Elements

The `dataset-name` component configuration element has no child elements.

D.30.2 Attributes

The `dataset-name` component has no attributes.

D.30.3 Example

The following example shows how to use the `dataset-name` element in the component configuration file:

```

<record-parameters>
  <dataset-name>tuple1</dataset-name>
  <event-type-list>
    <event-type>TupleEvent1</event-type>
  </event-type-list>
  <provider-name>test-rdbms-provider</provider-name>
  <batch-size>1</batch-size>
  <batch-time-out>10</batch-time-out>
</record-parameters>

```

D.31 delivery-mode

Use this element to define the delivery mode for a `jms-adapter`.

Valid values are:

- `persistent` (default)
- `nonpersistent`

D.31.1 Child Elements

The `delivery-mode` component configuration element has no child elements.

D.31.2 Attributes

The `delivery-mode` component has no attributes.

D.31.3 Example

The following example shows how to use the `delivery-mode` element in the component configuration file:

```
<jms-adapter>
  <name>jmsOutbound-map</name>
  <event-type>JMSTestEvent</event-type>
  <jndi-provider-url>t3://localhost:7001</jndi-provider-url>
  <destination-jndi-name>Topic1</destination-jndi-name>
  <delivery-mode>nonpersistent</delivery-mode>
</jms-adapter>
```

D.32 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.

D.32.1 Child Elements

The `destination-jndi-name` component configuration element has no child elements.

D.32.2 Attributes

The `destination-jndi-name` component has no attributes.

D.32.3 Example

The following example shows how to use the `destination-jndi-name` element in the component configuration file:

```
<jms-adapter>
  <name>jmsOutbound-map</name>
  <event-type>JMSTestEvent</event-type>
  <jndi-provider-url>t3://localhost:7001</jndi-provider-url>
  <destination-jndi-name>Topic1</destination-jndi-name>
  <delivery-mode>nonpersistent</delivery-mode>
</jms-adapter>
```

D.33 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.

D.33.1 Child Elements

The `destination-name` component configuration element has no child elements.

D.33.2 Attributes

The `destination-name` component has no attributes.

D.33.3 Example

The following example shows how to use the `destination-name` element in the component configuration file:

```
<jms-adapter>
  <name>jmsInbound-text</name>
  <connection-jndi-name>weblogic.jms.ConnectionFactory</connection-jndi-name>
  <destination-name>JMSServer-0/Module1!Queue1</destination-name>
```



```

<user>weblogic</user>
<password>weblogic</password>
<work-manager>JettyWorkManager</work-manager>
<concurrent-consumers>1</concurrent-consumers>
<session-transacted>>false</session-transacted>
</jms-adapter>

```

D.34 diagnostic-profiles

Use this element to define one or more Oracle CEP diagnostic profiles.

For more information, see "Monitoring the Throughput and Latency of a Stage or Path in the EPN" in the *Oracle Complex Event Processing Visualizer User's Guide*.

D.34.1 Child Elements

The `diagnostics-profiles` component configuration element supports the following child elements:

- [name](#)
- [profile](#)

D.34.2 Attributes

The `diagnostics-profiles` component has no attributes.

D.34.3 Example

The following example shows how to use the `diagnostics-profiles` element in the component configuration file:

```

<diagnostics-profiles>
  <name>myDiagnosticProfiles</name>
  <profile>
    ...
  </profile>
</diagnostics-profiles>

```

In the example, the channel's unique identifier is `myDiagnosticProfiles`.

D.35 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 Complex Event Processing Visualizer User's Guide*.

D.35.1 Child Elements

The `direction` component configuration has no child elements:

D.35.2 Attributes

The `direction` component has no attributes.

D.35.3 Example

The following example shows how to use the `direction` element in the component configuration file:

```
<diagnostic-profiles>
  <name>myselfprofiles</name>
  <profile>
    <name>testProfile0</name>
    <enabled>true</enabled>
    <start-stage>MetricSubscriber</start-stage>
    <max-latency>
      <collect-interval>
        <amount>1000</amount>
        <unit>s</unit>
      </collect-interval>
      <name>testProfile0MaxLat</name>
      <start-location>
        <application>diagnostic</application>
        <stage>MetricSubscriber</stage>
        <direction>INBOUND</direction>
      </start-location>
      <end-location>
        <application>diagnostic</application>
        <stage>MonitorProcessor</stage>
        <direction>OUTBOUND</direction>
      </end-location>
    </max-latency>
  </profile>
</diagnostic-profiles>
```

D.36 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.

D.36.1 Child Elements

The `duration` component configuration element has no child elements.

D.36.2 Attributes

The `duration` component has no attributes.

D.36.3 Example

The following example shows how to use the `duration` element in the component configuration file:

```
<record-parameters>
  <dataset-name>tuple1</dataset-name>
  <event-type-list>
    <event-type>TupleEvent1</event-type>
```

```

</event-type-list>
<provider-name>test-rdbms-provider</provider-name>
<store-policy-parameters>
  <parameter>
    <name>timeout</name>
    <value>300</value>
  </parameter>
</store-policy-parameters>
<time-range-offset>
  <start>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>

```

D.37 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 Complex Event Processing Visualizer User's Guide*.

D.37.1 Child Elements

The `enabled` component configuration element has no child elements.

D.37.2 Attributes

The `enabled` component has no attributes.

D.37.3 Example

The following example shows how to use the `enabled` element in the component configuration file:

```

<diagnostic-profiles>
  <name>myselfprofiles</name>
  <profile>
    <name>testProfile0</name>
    <enabled>true</enabled>
    <start-stage>MetricSubscriber</start-stage>
    <max-latency>
      <start-location>
        <application>diagnostic</application>
        <stage>MetricSubscriber</stage>
        <direction>INBOUND</direction>
      </start-location>
      <end-location>
        <application>diagnostic</application>
        <stage>MonitorProcessor</stage>
        <direction>OUTBOUND</direction>
      </end-location>
    </max-latency>
  </profile>
</diagnostic-profiles>

```

D.38 encrypted-password

Use the `encrypted-password` element in the following parent elements:

- `http-pub-sub-adapter`: Use the `encrypted-password` element to define the encrypted password if the HTTP pub-sub server to which the Oracle CEP application is publishing requires user authentication.
- `jms-adapter`: When Oracle CEP acquires the JNDI `InitialContext`, it uses the `user` and `password` (or `encrypted-password`) elements. When Oracle CEP 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 CEP the `user` and `password` (or `encrypted-password`) elements.

Use either `encrypted-password` or `password` but not both.

For more information, see [Section 7.4, "Encrypting Passwords in the JMS Adapter Component Configuration File"](#).

D.38.1 Child Elements

The `encrypted-password` component configuration element has no child elements.

D.38.2 Attributes

The `encrypted-password` component has no attributes.

D.38.3 Example

The following example shows how to use the `encrypted-password` element in the component configuration file:

```
<http-pub-sub-adapter>
  <name>remotePub</name>
  <server-url>http://localhost:9002/pubsub</server-url>
  <channel>/test1</channel>
  <event-type>com.bea.wlevs.tests.httppubsub.PubsubTestEvent</event-type>
  <user>wlevs</user>
  <encrypted-password>{Salted-3DES}s4YUEvH4Wl2DAjb45iJnrw==</encrypted-password>
</http-pub-sub-adapter>
```

D.39 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>.

D.39.1 Child Elements

The end component configuration element has no child elements.

D.39.2 Attributes

The end component has no attributes.

D.39.3 Example

The following example shows how to use the end element in the component configuration file:

```
<record-parameters>
  <dataset-name>tuple1</dataset-name>
  <event-type-list>
    <event-type>TupleEvent1</event-type>
  </event-type-list>
  <provider-name>test-rdbms-provider</provider-name>
  <store-policy-parameters>
    <parameter>
      <name>timeout</name>
      <value>300</value>
    </parameter>
  </store-policy-parameters>
  <time-range>
    <start>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>
```

D.40 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 Complex Event Processing Visualizer User's Guide*.

D.40.1 Child Elements

The end-location component configuration element supports the following child elements:

- [application](#)
- [stage](#)
- [direction](#)

D.40.2 Attributes

The end-location component has no attributes.

D.40.3 Example

The following example shows how to use the `end-location` element in the component configuration file:

```
<diagnostic-profiles>
  <name>myselfprofiles</name>
  <profile>
    <name>testProfile0</name>
    <enabled>true</enabled>
    <start-stage>MetricSubscriber</start-stage>
    <max-latency>
      <collect-interval>
        <amount>1000</amount>
        <unit>s</unit>
      </collect-interval>
      <name>testProfile0MaxLat</name>
      <start-location>
        <application>diagnostic</application>
        <stage>MetricSubscriber</stage>
        <direction>INBOUND</direction>
      </start-location>
      <end-location>
        <application>diagnostic</application>
        <stage>MonitorProcessor</stage>
        <direction>OUTBOUND</direction>
      </end-location>
    </max-latency>
  </profile>
</diagnostic-profiles>
```

D.41 event-bean

Use this element to define an event bean component.

For more information, see [Chapter 15, "Configuring Custom Event Beans"](#).

D.41.1 Child Elements

The `event-bean` component configuration element supports the following child elements:

- [name](#)
- [record-parameters](#)
- [playback-parameters](#)

D.41.2 Attributes

The `event-bean` component has no attributes.

D.41.3 Example

The following example shows how to use the `event-bean` element in the component configuration file:

```
<event-bean>
  <name>myEventBean</name>
</event-bean>
```

In the example, the channel's unique identifier is `myEventBean`.

D.42 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 CEP 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 2.2, "Creating an Oracle CEP 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 CEP to automatically perform the conversion between JMS messages and events. If you have created your own custom converter bean, then do not specify this element.
- [record-parameters](#): Use the `event-type` element to specify an event that you want to record.

The value of the `event-type` element must be one of the event types you defined in your event type repository. For more information, see [Section 1.1.2, "Oracle CEP Event Types"](#).

D.42.1 Child Elements

The `event-type` component configuration element has no child elements.

D.42.2 Attributes

The `event-type` component has no attributes.

D.42.3 Example

The following example shows how to use the `event-type` element in the component configuration file:

```
<record-parameters>
  <dataset-name>tuple1</dataset-name>
  <event-type-list>
    <event-type>TupleEvent1</event-type>
  </event-type-list>
  <provider-name>test-rdbms-provider</provider-name>
  <batch-size>1</batch-size>
  <batch-time-out>10</batch-time-out>
</record-parameters>
```

D.43 event-type-list

Use this element to define one or more events for record or playback for a component.

D.43.1 Child Elements

The `event-type-list` component configuration element supports the following child elements:

- `event-type`

D.43.2 Attributes

The `event-type-list` component has no attributes.

D.43.3 Example

The following example shows how to use the `event-type-list` element in the component configuration file:

```
<record-parameters>
  <dataset-name>tuple1</dataset-name>
  <event-type-list>
    <event-type>TupleEvent1</event-type>
  </event-type-list>
  <provider-name>test-rdbms-provider</provider-name>
  <batch-size>1</batch-size>
  <batch-time-out>10</batch-time-out>
</record-parameters>
```

D.44 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

D.44.1 Child Elements

The `eviction-policy` component configuration element has no child elements.

D.44.2 Attributes

The `eviction-policy` component has no attributes.

D.44.3 Example

The following example shows how to use the `eviction-policy` element in the component configuration file:

```
<caching-system>
  <name>providerCachingSystem</name>
  <cache>
    <name>providerCache</name>
    <max-size>1000</max-size>
    <eviction-policy>FIFO</eviction-policy>
    <time-to-live>60000</time-to-live>
    <idle-time>120000</idle-time>
    <write-none/>
  </cache>
</caching-system>
```



```

    <work-manager-name>JettyWorkManager</work-manager-name>
    <listeners asynchronous="false">
      <work-manager-name>JettyWorkManager</work-manager-name>
    </listeners>
  </cache>
</caching-system>

```

D.45 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 D.11, "bindings \(jms-adapter\)"](#).

D.45.1 Child Elements

The `group-binding` component configuration element supports the following child elements:

- `param`

D.45.2 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.

D.45.3 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>

```

```
</jms-adapter>
```

In this configuration, when the application is deployed to an Oracle CEP 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.

D.46 heartbeat

Use this element to define a heartbeat timeout for a system-timestamped `channel` component.

For system-timestamped relations or streams, time is dependent upon the arrival of data on the relation or stream data source. Oracle CEP generates a heartbeat on a system timestamped relation or stream if there is no activity (no data arriving on the stream or relation's source) for more than this number of nanoseconds. Either the relation or stream is populated by its specified source or Oracle CEP generates a heartbeat every `heartbeat` 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.

D.46.1 Child Elements

The `heartbeat` component configuration element has no child elements.

D.46.2 Attributes

The `heartbeat` component configuration element has no attributes.

D.46.3 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`.

D.47 http-pub-sub-adapter

Use this element to define an HTTP publish-subscribe server adapter component.

For more information, see [Chapter 8, "Configuring HTTP Publish-Subscribe Server Adapters"](#).

D.47.1 Child Elements

The `http-pub-sub-adapter` component configuration element supports the following child elements:

- `name`
- `record-parameters`
- `playback-parameters`
- `symbols`
- `work-manager-name`
- `netio`
- One of:
 - `server-context-path`
 - `server-url`
- `event-type`
- `user`
- One of:
 - `password`
 - `encrypted-password`

D.47.2 Attributes

The `http-pub-sub-adapter` component configuration element has no attributes.

D.47.3 Example

The following example shows how to use the `http-pub-sub-adapter` element in the component configuration file:

```
<http-pub-sub-adapter>
  <name>remotePub</name>
  <server-url>http://localhost:9002/pubsub</server-url>
  <channel>/test1</channel>
  <event-type>com.bea.wlevs.tests.httppubsub.PubsubTestEvent</event-type>
  <user>wlevs</user>
  <password>wlevs</password>
</http-pub-sub-adapter>
```

In the example, the adapter's unique identifier is `remotePub`.

D.48 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.

D.48.1 Child Elements

The `idle-time` component configuration element has no child elements.

D.48.2 Attributes

The `idle-time` component has no attributes.

D.48.3 Example

The following example shows how to use the `idle-time` element in the component configuration file:

```
<catching-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>
</catching-system>
```

D.49 jms-adapter

Use this element to define a JMS adapter component.

For more information, see [Chapter 7, "Configuring JMS Adapters"](#).

D.49.1 Child Elements

The `jms-adapter` component configuration element supports the following child elements:

- `name`
- `record-parameters`
- `playback-parameters`
- `event-type`
- `jndi-provider-url`
- `connection-jndi-name`
- One of:
 - `destination-jndi-name`
 - `destination-name`
- `user`
 - One of:
 - `password`
 - `encrypted-password`
- `connection-user`
 - One of:
 - `connection-password`
 - `connection-encrypted-password`
- `work-manager`

- `concurrent-consumers`
- `message-selector`
- `session-ack-mode-name`
- `session-transacted`
- `delivery-mode`

D.49.2 Attributes

The `.jms-adapter` component configuration element has no attributes.

D.49.3 Example

The following example shows how to use the `.jms-adapter` element in the component configuration file:

```
<jms-adapter>
  <name>jmsInbound-text</name>
  <jndi-provider-url>t3://localhost:7001</jndi-provider-url>
  <destination-name>JMSServer-0/Module1!Queue1</destination-name>
  <user>weblogic</user>
  <password>weblogic</password>
  <work-manager>JettyWorkManager</work-manager>
  <concurrent-consumers>1</concurrent-consumers>
  <session-transacted>>false</session-transacted>
</jms-adapter>
```

In the example, the adapter's unique identifier is `jmsInbound-text`.

D.50 jndi-factory

Use this optional element to define a JNDI factory for a `.jms-adapter`. The JNDI factory name. Default value is `weblogic.jndi.WLInitialContextFactory`, for Oracle CEP server JMS

D.50.1 Child Elements

The `jndi-factory` component configuration element has no child elements.

D.50.2 Attributes

The `jndi-factory` component has no attributes.

D.50.3 Example

The following example shows how to use the `jndi-provider-url` element in the component configuration file:

```
<jms-adapter>
  <name>jmsInbound-text</name>
  <jndi-factory>weblogic.jndi.WLInitialContextFactory</jndi-factory>
  <destination-name>JMSServer-0/Module1!Queue1</destination-name>
  <user>weblogic</user>
  <password>weblogic</password>
  <work-manager>JettyWorkManager</work-manager>
  <concurrent-consumers>1</concurrent-consumers>
  <session-transacted>>false</session-transacted>
</jms-adapter>
```

D.51 jndi-provider-url

Use this required element to define a JNDI provider URL for a `jms-adapter`.

D.51.1 Child Elements

The `jndi-provider-url` component configuration element has no child elements.

D.51.2 Attributes

The `jndi-provider-url` component has no attributes.

D.51.3 Example

The following example shows how to use the `jndi-provider-url` element in the component configuration file:

```
<jms-adapter>
  <name>jmsInbound-text</name>
  <jndi-provider-url>t3://localhost:7001</jndi-provider-url>
  <destination-name>JMSServer-0/Module1!Queue1</destination-name>
  <user>weblogic</user>
  <password>weblogic</password>
  <work-manager>JettyWorkManager</work-manager>
  <concurrent-consumers>1</concurrent-consumers>
  <session-transacted>>false</session-transacted>
</jms-adapter>
```

D.52 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.

D.52.1 Child Elements

The `listeners` component configuration element supports the following child elements:

- `work-manager-name`

D.52.2 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 <code>true</code> and <code>false</code> . Default value is <code>false</code> .	Boolean	No.

D.52.3 Example

The following example shows how to use the `listeners` element in the component configuration file:

```

<キャッシング-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>
</キャッシング-system>

```

D.53 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 Complex Event Processing Visualizer User's Guide*.

D.53.1 Child Elements

The `location` component configuration element supports the following child elements:

- [application](#)
- [stage](#)
- [direction](#)

D.53.2 Attributes

The `location` component has no attributes.

D.53.3 Example

The following example shows how to use the `location` element in the component configuration file:

```

<diagnostic-profiles>
  <name>myselfprofiles</name>
  <profile>
    <name>testProfile0</name>
    <enabled>true</enabled>
    <start-stage>MetricSubscriber</start-stage>
    <throughput>
      <throughput-interval>
        <amount>100000</amount>
        <unit>MICROSECONDS</unit>
      </throughput-interval>
      <average-interval>
        <amount>100000000</amount>
      </average-interval>
    </throughput>
  </profile>
</diagnostic-profiles>

```

```
        <unit>NANOSECONDS</unit>
    </average-interval>
    <location>
        <application>diagnostic</application>
        <stage>AlertEventStream</stage>
        <direction>INBOUND</direction>
    </location>
</throughput>
</profile>
</diagnostic-profiles>
```

D.54 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 Complex Event Processing Visualizer User's Guide*.

D.54.1 Child Elements

The `max-latency` component configuration element supports the following child elements:

- `name`
- `collect-interval`
- `start-location`
- `end-location`

D.54.2 Attributes

The `max-latency` component has no attributes.

D.54.3 Example

The following example shows how to use the `max-latency` element in the component configuration file:

```
<diagnostic-profiles>
  <name>myselfprofiles</name>
  <profile>
    <name>testProfile0</name>
    <enabled>true</enabled>
    <start-stage>MetricSubscriber</start-stage>
    <max-latency>
      <start-location>
        <application>diagnostic</application>
        <stage>MetricSubscriber</stage>
        <direction>INBOUND</direction>
      </start-location>
      <end-location>
        <application>diagnostic</application>
        <stage>MonitorProcessor</stage>
        <direction>OUTBOUND</direction>
      </end-location>
    </max-latency>
  </profile>
</diagnostic-profiles>
```


D.55 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

D.55.1 Child Elements

The `max-size` component configuration element has no child elements.

D.55.2 Attributes

The `max-size` component has no attributes.

D.55.3 Example

The following example shows how to use the `max-size` element in the component configuration file:

```
<stream>
  <name>monitoring-control-stream</name>
  <max-size>10000</max-size>
  <max-threads>1</max-threads>
</stream>
```

D.56 max-threads

Use this element to define the maximum number of threads that Oracle CEP server uses to process events for a `channel` or `stream`. The default value is 0..

When set to 0, the channel acts as a pass-through. When `max-threads > 0`, the channel acts as classic blocking queue, where upstream components are producers of events and the downstream components are the consumers of events. The queue size is defined by the configuration `max-size`. There will be up to `max-threads` number of threads consuming events from the queue.

You can change `max-threads` from 0 to a positive integer (that is, from a pass through to multiple threads) without redeploying. However, if you change `max-threads` from a positive integer to 0 (that is, from multiple threads to a pass through), then you must redeploy your application.

If the `max-size` attribute is 0, then setting a value for `max-threads` has no effect.

The default value for this attribute is 0.

Setting this value has no effect when `max-size` is 0.

D.56.1 Child Elements

The `max-threads` component configuration element has no child elements.

D.56.2 Attributes

The `max-threads` component has no attributes.

D.56.3 Example

The following example shows how to use the `max-threads` element in the component configuration file:

```
<channel>
  <name>monitoring-control-stream</name>
  <max-size>10000</max-size>
  <max-threads>1</max-threads>
</channel>
```

D.57 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'
```

D.57.1 Child Elements

The `message-selector` component configuration element has no child elements.

D.57.2 Attributes

The `message-selector` component has no attributes.

D.57.3 Example

The following example shows how to use the `message-selector` element in the component configuration file:

```
<jms-adapter>
  <name>jmsInbound-text</name>
  <connection-jndi-name>weblogic.jms.ConnectionFactory</connection-jndi-name>
  <destination-name>JMSServer-0/Module1!Queue1</destination-name>
  <user>weblogic</user>
  <password>weblogic</password>
  <work-manager>JettyWorkManager</work-manager>
  <message-selector>EventType = 'News' OR 'Commentary'</message-selector>
  <session-transacted>>false</session-transacted>
</jms-adapter>
```

D.58 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.

D.58.1 Child Elements

The name component configuration element has no child elements:

D.58.2 Attributes

The name component has no attributes.

D.58.3 Example

The following example shows how to use the name element in the component configuration file:

```
<diagnostics-profiles>
  <name>myDiagnosticProfiles</name>
  <profile>
    ...
  </profile>
</diagnostics-profiles>
```

In the example, the channel's unique identifier is `myDiagnosticProfiles`.

D.59 netio

Use this element to define a network input/output port for a component.

D.59.1 Child Elements

The `netio` component configuration element supports the following child elements:

- `provider-name`
- `num-threads`
- `accept-backlog`

D.59.2 Attributes

The `netio` component has no attributes.

D.59.3 Example

The following example shows how to use the `netio` element in the component configuration file:

```
<netio>
  <provider-name>providerCache</provider-name>
  <num-threads>1000</num-threads>
</netio>
```

D.60 num-threads

Use this element to define the number of threads in a network input/output port for a component.

D.60.1 Child Elements

The `num-threads` component configuration element has no child elements.

D.60.2 Attributes

The `num-threads` component has no attributes.

D.60.3 Example

The following example shows how to use the `num-threads` element in the component configuration file:

```
<netio>
  <provider-name>providerCache</provider-name>
  <num-threads>1000</num-threads>
</netio>
```

D.61 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 D.11, "bindings \(jms-adapter\)"](#).

D.61.1 Child Elements

The `param` component configuration element has no child elements.

D.61.2 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?
<code>id</code>	The parameter name specified in the <code>message-selector</code> .	String	Yes.

D.61.3 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>
  </bindings>
```

```

    <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 CEP 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.

D.62 parameter

Use this element to define a name/value parameter for a component.

D.62.1 Child Elements

The `parameter` component configuration element supports the following child elements:

- `name`
- `value`

D.62.2 Attributes

The `parameter` component has no attributes.

D.62.3 Example

The following example shows how to use the `parameter` element in the component configuration file:

```

<record-parameters>
  <dataset-name>tuple1</dataset-name>
  <event-type-list>
    <event-type>TupleEvent1</event-type>
  </event-type-list>
  <provider-name>test-rdbms-provider</provider-name>
  <store-policy-parameters>
    <parameter>
      <name>timeout</name>
      <value>300</value>
    </parameter>
  </store-policy-parameters>
  <batch-size>1</batch-size>
  <batch-time-out>10</batch-time-out>
</record-parameters>

```

D.63 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 Complex Event Processing CQL Language Reference*
- "Parameterized Queries" in the *Oracle Complex Event Processing EPL Language Reference*

D.63.1 Child Elements

The `params` component configuration element has no child elements.

D.63.2 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.

D.63.3 Example

The following example shows how to use the `params` element in the component configuration file:

```
<processor>
  <name>processor1</name>
  <record-parameters>
    <dataset-name>test1data</dataset-name>
    <event-type-list>
      <event-type>SimpleEvent</event-type>
    </event-type-list>
    <provider-name>test-rdbms-provider</provider-name>
    <batch-size>1</batch-size>
    <batch-time-out>10</batch-time-out>
  </record-parameters>
  <rules>
    <rule id="rule1"><![CDATA[
      select stockSymbol, avg(price) as percentage
      from StockTick retain 5 events
      where stockSymbol=?
      having avg(price) > ? or avg(price) < ?
    ]]></rule>
  </rules>
  <bindings>
    <binding id="rule1">
      <params>BEAS,10.0,-10.0</params>
      <params id="IBM">IBM,5.0,5.0</params>
    </binding>
  </bindings>
</processor>
```

D.64 password

Use the `password` element in the following parent elements:

- [http-pub-sub-adapter](#): Use the `password` element to define the user password if the HTTP pub-sub server to which the Oracle CEP application is publishing requires user authentication.
- [jms-adapter](#): When Oracle CEP acquires the JNDI `InitialContext`, it uses the `user` and `password` (or [encrypted-password](#)) elements. When Oracle CEP 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 CEP the `user` and `password` (or `encrypted-password`) elements.

Use either `encrypted-password` or `password` but not both.

D.64.1 Child Elements

The `password` component configuration element has no child elements.

D.64.2 Attributes

The `password` component has no attributes.

D.64.3 Example

The following example shows how to use the `password` element in the component configuration file:

```
<http-pub-sub-adapter>
  <name>remotePub</name>
  <server-url>http://localhost:9002/pubsub</server-url>
  <channel>/test1</channel>
  <event-type>com.bea.wlevs.tests.httppubsub.PubsubTestEvent</event-type>
  <user>wlevs</user>
  <password>wlevs</password>
</http-pub-sub-adapter>
```

D.65 playback-parameters

Use this element to define event playback parameters for a component.

For more information, see [Chapter 13, "Configuring Event Record and Playback"](#).

D.65.1 Child Elements

The `playback-parameters` component configuration element supports the following child elements:

- `dataset-name`
- `event-type-list`
- `provider-name`
- `store-policy-parameters`
- `max-size`
- `max-threads`
- One of:
 - `time-range`
 - `time-range-offset`
- One of:
 - `schedule-time-range`
 - `schedule-time-range-offset`
- `playback-speed`

- [repeat](#)

D.65.2 Attributes

The `playback-parameters` component has no attributes.

D.65.3 Example

The following example shows how to use the `playback-parameters` element in the component configuration file:

```
<playback-parameters>
  <dataset-name>tuple1</dataset-name>
  <event-type-list>
    <event-type>TupleEvent1</event-type>
  </event-type-list>
  <provider-name>test-rdbms-provider</provider-name>
</playback-parameters>
```

D.66 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.

D.66.1 Child Elements

The `playback-speed` component configuration element has no child elements.

D.66.2 Attributes

The `playback-speed` component has no attributes.

D.66.3 Example

The following example shows how to use the `duration` element in the component configuration file:

```
<playback-parameters>
  <dataset-name>tuple1</dataset-name>
  <event-type-list>
    <event-type>TupleEvent1</event-type>
  </event-type-list>
  <provider-name>test-rdbms-provider</provider-name>
  <store-policy-parameters>
    <parameter>
      <name>timeout</name>
      <value>300</value>
    <parameter>
  </store-policy-parameters>
  <time-range-offset>
    <start>2010-01-20T05:00:00</start>
    <duration>03:00:00</duration>
  </time-range-offset>
  <playback-speed>100</playback-speed>
</playback-parameters>
```


D.67 processor (EPL)

Use this element to define an Oracle CQL or EPL processor component.

For more information, see [Chapter 11, "Configuring EPL Processors"](#).

For information on the processor element for Oracle CQL processors, see [processor \(Oracle CQL\)](#).

D.67.1 Child Elements

The `processor` component configuration element supports the following child elements:

- [name](#)
- [record-parameters](#)
- [playback-parameters](#)
- [rules](#)
- [database](#)
- [bindings \(processor\)](#)

D.67.2 Attributes

The `processor` component has no attributes.

D.67.3 Example

The following example shows how to use the `processor` element in the component configuration file:

```
<processor>
  <name>processor1</name>
  <record-parameters>
    <dataset-name>test1data</dataset-name>
    <event-type-list>
      <event-type>SimpleEvent</event-type>
    </event-type-list>
    <provider-name>test-rdbms-provider</provider-name>
    <batch-size>1</batch-size>
    <batch-time-out>10</batch-time-out>
  </record-parameters>
  <rules>
    <rule id="rule1"><![CDATA[
      select stockSymbol, avg(price) as percentage
      from StockTick retain 5 events
      where stockSymbol=?
      having avg(price) > ? or avg(price) < ?
    ]]></rule>
  </rules>
  <bindings>
    <binding id="rule1">
      <params>BEAS,10.0,-10.0</params>
      <params id="IBM">IBM,5.0,5.0</params>
    </binding>
  </bindings>
</processor>
```

In the example, the processor's unique identifier is `processor1`.

D.68 processor (Oracle CQL)

Use this element to define an Oracle CQL processor component.

For more information, see [Chapter 10, "Configuring Oracle CQL Processors"](#).

For information on the processor element for EPL processors, see [processor \(EPL\)](#).

D.68.1 Child Elements

The `processor` component configuration element supports the following child elements:

- [name](#)
- [record-parameters](#)
- [playback-parameters](#)
- [rules](#)
- [bindings \(processor\)](#)

D.68.2 Attributes

The processor component has no attributes.

D.68.3 Example

The following example shows how to use the `processor` element in the component configuration file:

```
<processor>
  <name>cqlProcessor</name>
  <rules>
    <view id="lastEvents" schema="cusip bid srcId bidQty ask askQty seq"><![CDATA[
      select cusip, bid, srcId, bidQty, ask, askQty, seq
      from inputChannel[partition by srcId, cusip rows 1]
    ]]></view>
    <view id="bidask" schema="cusip bid ask"><![CDATA[
      select cusip, max(bid), min(ask)
      from lastEvents
      group by cusip
    ]]></view>
    <view ...><![CDATA[
      ...
    ]]></view>
    ...
    <view id="MAXBIDMINASK" schema="cusip bidseq bidSrcId bid askseq askSrcId ask bidQty
askQty"><![CDATA[
      select bid.cusip, bid.seq, bid.srcId as bidSrcId, bid.bid, ask.seq,
      ask.srcId as askSrcId, ask.ask, bid.bidQty, ask.askQty
      from BIDMAX as bid, ASKMIN as ask
      where bid.cusip = ask.cusip
    ]]></view>
    <query id="BBAQuery"><![CDATA[
      ISTREAM(select bba.cusip, bba.bidseq, bba.bidSrcId, bba.bid, bba.askseq,
      bba.askSrcId, bba.ask, bba.bidQty, bba.askQty,
      "BBAStrategy" as intermediateStrategy, p.seq as correlationId, 1 as priority
      from MAXBIDMINASK as bba, inputChannel[rows 1] as p where bba.cusip = p.cusip)
    ]]></query>
    <query id="MarketRule"><![CDATA[
      SELECT symbol, AVG(price) AS average, :1 AS market
      FROM StockTick [RANGE 5 SECONDS]
    ]]></query>
  </rules>
</processor>
```

```

        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`.

D.69 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 Complex Event Processing Visualizer User's Guide*.

D.69.1 Child Elements

The `profile` component configuration element supports the following child elements:

- `name`
- `enabled`
- `start-stage`
- `max-latency`
- `average-latency`
- `throughput`

D.69.2 Attributes

The `profile` component has no attributes.

D.69.3 Example

The following example shows how to use the `profile` element in the component configuration file:

```

<diagnostic-profiles>
  <name>myselfprofiles</name>
  <profile>
    <name>testProfile0</name>
    <enabled>true</enabled>
    <start-stage>MetricSubscriber</start-stage>
    <max-latency>
      <start-location>
        <application>diagnostic</application>
        <stage>MetricSubscriber</stage>
        <direction>INBOUND</direction>
      </start-location>
      <end-location>
        <application>diagnostic</application>
        <stage>MonitorProcessor</stage>
        <direction>OUTBOUND</direction>
      </end-location>
    </max-latency>
  </profile>
</diagnostic-profiles>

```

```
</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>
```

D.70 provider-name

Use the `provider-name` element in the following parent elements:

- **netio**: Use the `provider-name` element to define which provider to use for the underlying socket implementation. Valid value is an Oracle CEP server `config.xml` file `netio` child element `provider-type`.
- **record-parameters**: Use the `provider-name` element to define the name of the event store provider. The value of this element corresponds to the value of the `name` child element of the `rdbsms-event-store-provider` element in the `config.xml` file of the Oracle CEP server instance.

When configuring the Oracle RDBMS-based provider, you are required to specify this element.

This may be left blank to configure to use the default Berkeley database provider.

D.70.1 Child Elements

The `provider-name` component configuration element has no child elements.

D.70.2 Attributes

The `provider-name` component has no attributes.

D.70.3 Example

The following example shows how to use the `provider-name` element in the component configuration file:

```
<netio>
  <provider-name>providerCache</provider-name>
  <num-threads>1000</num-threads>
</netio>
```

D.71 query

Use this element to define an Oracle CQL query for a component.

For more information, see [Chapter 10, "Configuring Oracle CQL Processors"](#).

D.71.1 Child Elements

The `query` component configuration element has no child elements.

D.71.2 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.

D.71.3 Example

The following example shows how to use the `query` element in the component configuration file:

```
<processor>
  <name>cqlProcessor</name>
  <rules>
    <view id="lastEvents" schema="cusip bid srcId bidQty ask askQty seq"><![CDATA[
      select cusip, bid, srcId, bidQty, ask, askQty, seq
      from inputChannel[partition by srcId, cusip rows 1]
    ]]></view>
    <view id="bidask" schema="cusip bid ask"><![CDATA[
      select cusip, max(bid), min(ask)
      from lastEvents
      group by cusip
    ]]></view>
    <view ...><![CDATA[
      ...
    ]]></view>
    ...
    <view id="MAXBIDMINASK" schema="cusip bidseq bidSrcId bid askseq askSrcId ask bidQty askQty"><![CDATA[
      select bid.cusip, bid.seq, bid.srcId as bidSrcId, bid.bid, ask.seq,
      ask.srcId as askSrcId, ask.ask, bid.bidQty, ask.askQty
      from BIDMAX as bid, ASKMIN as ask
      where bid.cusip = ask.cusip
    ]]></view>
    <query id="BBAQuery"><![CDATA[
      ISTREAM(select bba.cusip, bba.bidseq, bba.bidSrcId, bba.bid, bba.askseq,
```

```
        bba.askSrcId, bba.ask, bba.bidQty, bba.askQty, "BBAStrategy" as
intermediateStrategy,
        p.seq as correlationId, 1 as priority
    from MAXBIDMINASK as bba, inputChannel[rows 1] as p where bba.cusip = p.cusip
  ]]></query>
</rules>
</processor>
```

D.72 record-parameters

Use this element to define event record parameters for a component.

For more information, see [Chapter 13, "Configuring Event Record and Playback"](#).

D.72.1 Child Elements

The record-parameters component configuration element supports the following child elements:

- [dataset-name](#)
- [event-type-list](#)
- [provider-name](#)
- [store-policy-parameters](#)
- [max-size](#)
- [max-threads](#)
- One of:
 - [time-range](#)
 - [time-range-offset](#)
- [batch-size](#)
- [batch-time-out](#)

D.72.2 Attributes

The record-parameters component has no attributes.

D.72.3 Example

The following example shows how to use the record-parameters element in the component configuration file:

```
<record-parameters>
  <dataset-name>tuple1</dataset-name>
  <event-type-list>
    <event-type>TupleEvent1</event-type>
  </event-type-list>
  <provider-name>test-rdbms-provider</provider-name>
  <batch-size>1</batch-size>
  <batch-time-out>10</batch-time-out>
</record-parameters>
```

D.73 repeat

Use this element to define whether or not to repeat [playback-parameters](#).

Valid values are:

- true
- false

D.73.1 Child Elements

The `repeat` component configuration element has no child elements.

D.73.2 Attributes

The `repeat` component has no attributes.

D.73.3 Example

The following example shows how to use the `duration` element in the component configuration file:

```
<playback-parameters>
  <dataset-name>tuple1</dataset-name>
  <event-type-list>
    <event-type>TupleEvent1</event-type>
  </event-type-list>
  <provider-name>test-rdbms-provider</provider-name>
  <store-policy-parameters>
    <parameter>
      <name>timeout</name>
      <value>300</value>
    </parameter>
  </store-policy-parameters>
  <time-range-offset>
    <start>2010-01-20T05:00:00</start>
    <duration>03:00:00</duration>
  </time-range-offset>
  <repeat>true</repeat>
</playback-parameters>
```

D.74 rule

Use this element to define an EPL rule for a component.

This element is applicable only in a `rules` element.

D.74.1 Child Elements

The `rule` component configuration element has no child elements.

D.74.2 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?
<code>id</code>	Unique identifier for this rule.	String	Yes.
<code>active</code>	Execute this rule 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.

D.74.3 Example

The following example shows how to use the `rule` element in the component configuration file:

```
<processor>
  <name>rvSampleProcessor</name>
  <rules>
    <rule id="rvSampleRule1"><![CDATA[
      select * from RVSampleEvent retain 1 event
    ]]></rule>
  </rules>
</processor>
```

D.75 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 \(EPL\)](#).

D.75.1 Child Elements

The `rules` component configuration element supports the following child elements:

- [rule](#)
- [query](#)
- [view](#)

D.75.2 Attributes

The `rules` component has no attributes.

D.75.3 Example

The following example shows how to use the `rules` element in the component configuration file:

```
<processor>
  <name>cqlProcessor</name>
  <rules>
    <view id="lastEvents" schema="cusip bid srcId bidQty ask askQty seq"><![CDATA[
      select cusip, bid, srcId, bidQty, ask, askQty, seq
      from inputChannel[partition by srcId, cusip rows 1]
    ]]></view>
    <view id="bidask" schema="cusip bid ask"><![CDATA[
      select cusip, max(bid), min(ask)
      from lastEvents
      group by cusip
    ]]></view>
    <view ...><![CDATA[
      ...
    ]]></view>
    ...
    <view id="MAXBIDMINASK" schema="cusip bidseq bidSrcId bid askseq askSrcId ask bidQty askQty"><![CDATA[
      select bid.cusip, bid.seq, bid.srcId as bidSrcId, bid.bid, ask.seq,
      ask.srcId as askSrcId, ask.ask, bid.bidQty, ask.askQty
      from BIDMAX as bid, ASKMIN as ask
      where bid.cusip = ask.cusip
    ]]></view>
    <query id="BBAQuery"><![CDATA[
      ISTREAM(select bba.cusip, bba.bidseq, bba.bidSrcId, bba.bid, bba.askseq,
      bba.askSrcId, bba.ask, bba.bidQty, bba.askQty,
```



```

        "BBAStrategy" as intermediateStrategy, p.seq as correlationId, 1 as priority
        from MAXBIDMINASK as bba, inputChannel[rows 1] as p where bba.cusip = p.cusip)
    ]]></query>
</rules>
</processor>

```

D.76 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.

D.76.1 Child Elements

The `schedule-time-range` component configuration element supports the following child elements:

- `start`
- `end`

D.76.2 Attributes

The `schedule-time-range` component has no attributes.

D.76.3 Example

The following example shows how to use the `schedule-time-range` element in the component configuration file:

```

<record-parameters>
  <dataset-name>tuple1</dataset-name>
  <event-type-list>
    <event-type>TupleEvent1</event-type>
  </event-type-list>
  <provider-name>test-rdbms-provider</provider-name>
  <store-policy-parameters>
    <parameter>
      <name>timeout</name>
      <value>300</value>
    </parameter>
  </store-policy-parameters>
  <schedule-time-range>
    <start>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>

```

D.77 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.

D.77.1 Child Elements

The `schedule-time-range-offset` component configuration element supports the following child elements:

- `start`
- `duration`

D.77.2 Attributes

The `schedule-time-range-offset` component has no attributes.

D.77.3 Example

The following example shows how to use the `schedule-time-range-offset` element in the component configuration file:

```
<record-parameters>
  <dataset-name>tuple1</dataset-name>
  <event-type-list>
    <event-type>TupleEvent1</event-type>
  </event-type-list>
  <provider-name>test-rdbms-provider</provider-name>
  <store-policy-parameters>
    <parameter>
      <name>timeout</name>
      <value>300</value>
    </parameter>
  </store-policy-parameters>
  <schedule-time-range-offset>
    <start>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>
```

D.78 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 10, "Configuring Oracle CQL Processors"](#).

D.78.1 Child Elements

The `selector` component configuration element has no child elements.

D.78.2 Attributes

The `selector` component has no attributes.

D.78.3 Example

The following example shows how to use the `selector` element in the component configuration file:

```

<processor>
  <name>PatternProcessor</name>
  <rules>
    <query id="match"><![CDATA[
      select T.firstW as firstw, T.lastZ as lastz, T.price as price
      from StockInputsStream
      MATCH_RECOGNIZE (
        MEASURES A.c1 as firstW, last(Z.c1) as lastZ, A.c2 as price

```

```

        PATTERN(A W+ X+ Y+ Z+)
        DEFINE A as A.c1 = A.c1,
               W as W.c2 < prev(W.c2),
               X as X.c2 > prev(X.c2),
               Y as Y.c2 < prev(Y.c2),
               Z as Z.c2 > prev(Z.c2))
        as T
    ]]></query>
    <query id="stock"><![CDATA[
        select c1 as ts, c2 as price from StockInputsStream
    ]]></query>
</rules>
</processor>
<channel>
    <name>StockOutputChannel</name>
    <selector>stock</selector>
</channel>
<channel>
    <name>MatchOutputChannel</name>
    <selector>match</selector>
</channel>

```

D.79 server-context-path

Required. For each *local* [http-pub-sub-adapter](#) for publishing, specify the value of the Oracle CEP server `config.xml` file element `http-pubsub` child element `path` of the local HTTP pub-sub server associated with the Oracle CEP instance hosting the current Oracle CEP 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.

D.79.1 Child Elements

The `server-context-path` component configuration element has no child elements.

D.79.2 Attributes

The `server-context-path` component has no attributes.

D.79.3 Example

The following example shows how to use the `server-context-path` element in the component configuration file:

```

<http-pub-sub-adapter>
    <name>localPub</name>
    <server-context-path>/pubsub</server-context-path>
    <channel>/test1</channel>
</http-pub-sub-adapter>

```

D.80 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 CEP

application will publish. The remote HTTP pub-sub server could be another instance of Oracle CEP, or a WebLogic Server instance, or it could be any third-party HTTP pub-sub server. For example:

```
http://myhost.com:9102/pubsub
```

Note: Do not specify this option for a local HTTP pub-sub adapter.

D.80.1 Child Elements

The `server-url` component configuration element has no child elements.

D.80.2 Attributes

The `server-url` component has no attributes.

D.80.3 Example

The following example shows how to use the `server-url` element in the component configuration file:

```
<http-pub-sub-adapter>
  <name>remotePub</name>
  <server-url>http://myhost.com:9102/pubsub</server-url>
  <channel>/test1</channel>
  <event-type>com.bea.wlevs.tests.httppubsub.PubsubTestEvent</event-type>
  <user>wlevs</user>
  <password>wlevs</password>
</http-pub-sub-adapter>
```

In the example, the URL of the remote HTTP pub-sub server to which the `remotePublisher` adapter will publish events is `http://myhost.com:9102/pubsub`.

D.81 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`.

D.81.1 Child Elements

The `session-ack-mode-name` component configuration element has no child elements.

D.81.2 Attributes

The `session-ack-mode-name` component has no attributes.

D.81.3 Example

The following example shows how to use the `session-ack-mode-name` element in the component configuration file:

```
<jms-adapter>
  <name>jmsInbound-text</name>
  <connection-jndi-name>weblogic.jms.ConnectionFactory</connection-jndi-name>
  <destination-name>JMSServer-0/Module1!Queue1</destination-name>
  <user>weblogic</user>
  <password>weblogic</password>
  <work-manager>JettyWorkManager</work-manager>
  <session-ack-mode-name>AUTO_ACKNOWLEDGE</session-ack-mode-name>
  <session-transacted>>false</session-transacted>
</jms-adapter>
```

D.82 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

D.82.1 Child Elements

The `session-transacted` component configuration element has no child elements.

D.82.2 Attributes

The `session-transacted` component has no attributes.

D.82.3 Example

The following example shows how to use the `session-transacted` element in the component configuration file:

```
<jms-adapter>
  <name>jmsInbound-text</name>
  <connection-jndi-name>weblogic.jms.ConnectionFactory</connection-jndi-name>
  <destination-name>JMSServer-0/Module1!Queue1</destination-name>
  <user>weblogic</user>
  <password>weblogic</password>
  <work-manager>JettyWorkManager</work-manager>
  <session-ack-mode-name>AUTO_ACKNOWLEDGE</session-ack-mode-name>
  <session-transacted>>false</session-transacted>
</jms-adapter>
```

D.83 stage

Use this element to define the stage for a [start-location](#) or [end-location](#) element of a diagnostic profile.

Valid values are the name of an existing stage in your Event Processing Network (EPN). For more information, see [Section 1.1.1, "Components of the Oracle CEP Event Processing Network"](#).

D.83.1 Child Elements

The stage component configuration has no child elements:

D.83.2 Attributes

The stage component has no attributes.

D.83.3 Example

The following example shows how to use the `stage` element in the component configuration file:

```
<diagnostic-profiles>
  <name>myselfprofiles</name>
  <profile>
    <name>testProfile0</name>
    <enabled>true</enabled>
    <start-stage>MetricSubscriber</start-stage>
    <max-latency>
      <collect-interval>
        <amount>1000</amount>
        <unit>s</unit>
      </collect-interval>
      <name>testProfile0MaxLat</name>
      <start-location>
        <application>diagnostic</application>
        <stage>MetricSubscriber</stage>
        <direction>INBOUND</direction>
      </start-location>
      <end-location>
        <application>diagnostic</application>
        <stage>MonitorProcessor</stage>
        <direction>OUTBOUND</direction>
      </end-location>
    </max-latency>
  </profile>
</diagnostic-profiles>
```

D.84 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>.

D.84.1 Child Elements

The `start` component configuration element has no child elements.

D.84.2 Attributes

The `start` component has no attributes.

D.84.3 Example

The following example shows how to use the `start` element in the component configuration file:

```
<record-parameters>
  <dataset-name>tuple1</dataset-name>
  <event-type-list>
    <event-type>TupleEvent1</event-type>
  </event-type-list>
  <provider-name>test-rdbms-provider</provider-name>
  <store-policy-parameters>
    <parameter>
      <name>timeout</name>
      <value>300</value>
    </parameter>
  </store-policy-parameters>
  <time-range>
    <start>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>
```

D.85 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 Complex Event Processing Visualizer User's Guide*.

D.85.1 Child Elements

The `start-location` component configuration element supports the following child elements:

- [application](#)
- [stage](#)
- [direction](#)

D.85.2 Attributes

The `start-location` component has no attributes.

D.85.3 Example

The following example shows how to use the `start-location` element in the component configuration file:

```
<diagnostic-profiles>
  <name>myselfprofiles</name>
  <profile>
    <name>testProfile0</name>
    <enabled>true</enabled>
    <start-stage>MetricSubscriber</start-stage>
    <max-latency>
      <collect-interval>
        <amount>1000</amount>
        <unit>s</unit>
      </collect-interval>
      <name>testProfile0MaxLat</name>
      <start-location>
        <application>diagnostic</application>
        <stage>MetricSubscriber</stage>
        <direction>INBOUND</direction>
      </start-location>
      <end-location>
        <application>diagnostic</application>
        <stage>MonitorProcessor</stage>
        <direction>OUTBOUND</direction>
      </end-location>
    </max-latency>
  </profile>
</diagnostic-profiles>
```

D.86 start-stage

Use this element to define the starting stage of a diagnostic profile.

Valid values are the name of an existing stage in your Event Processing Network (EPN). For more information, see [Section 1.1.1, "Components of the Oracle CEP Event Processing Network"](#).

For more information, see "Monitoring the Throughput and Latency of a Stage or Path in the EPN" in the *Oracle Complex Event Processing Visualizer User's Guide*.

D.86.1 Child Elements

The `start-stage` component configuration element has no child elements.

D.86.2 Attributes

The `start-stage` component has no attributes.

D.86.3 Example

The following example shows how to use the `start-stage` element in the component configuration file:

```
<diagnostic-profiles>
  <name>myselfprofiles</name>
  <profile>
    <name>testProfile0</name>
    <enabled>true</enabled>
    <start-stage>MetricSubscriber</start-stage>
    <max-latency>
      <start-location>
```

```
        <application>diagnostic</application>
        <stage>MetricSubscriber</stage>
        <direction>INBOUND</direction>
    </start-location>
    <end-location>
        <application>diagnostic</application>
        <stage>MonitorProcessor</stage>
        <direction>OUTBOUND</direction>
    </end-location>
</max-latency>
</profile>
</diagnostic-profiles>
```

D.87 store-policy-parameters

Use this element to define one or more store policy parameter, specific to the event store provider.

D.87.1 Child Elements

The `store-policy-parameter` component configuration element supports the following child elements:

- `parameter`

D.87.2 Attributes

The `store-policy-parameter` component has no attributes.

D.87.3 Example

The following example shows how to use the `store-policy-parameter` element in the component configuration file:

```
<record-parameters>
  <dataset-name>tuple1</dataset-name>
  <event-type-list>
    <event-type>TupleEvent1</event-type>
  </event-type-list>
  <provider-name>test-rdbms-provider</provider-name>
  <store-policy-parameters>
    <parameter>
      <name>timeout</name>
      <value>300</value>
    </parameter>
  </store-policy-parameters>
  <batch-size>1</batch-size>
  <batch-time-out>10</batch-time-out>
</record-parameters>
```

D.88 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.

D.88.1 Child Elements

The `stream` component configuration element supports the following child elements:

- [name](#)
- [record-parameters](#)
- [playback-parameters](#)
- [max-size](#)
- [max-threads](#)

D.88.2 Attributes

The `stream` component has no attributes.

D.88.3 Example

The following example shows how to use the `stream` element in the component configuration file:

```
<stream>
  <name>fxMarketEuroOut</name>
  <max-size>0</max-size>
  <max-threads>0</max-threads>
</stream>
```

In the example, the stream's unique identifier is `fxMarketEuroOut`.

D.89 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).

D.89.1 Child Elements

The `symbol` component configuration has no child elements:

D.89.2 Attributes

The `symbol` component has no attributes.

D.89.3 Example

The following example shows how to use the `symbol` element in the component configuration file:

```
<adapter>
  <name>trackdata</name>
  <symbols>
    <symbol>BEAS</symbol>
    <symbol>IBM</symbol>
  </symbols>
</adapter>
```

D.90 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).

D.90.1 Child Elements

The `symbols` component configuration element supports the following child elements:

- `symbol`

D.90.2 Attributes

The `symbols` component has no attributes.

D.90.3 Example

The following example shows how to use the `symbols` element in the component configuration file:

```
<adapter>
  <name>trackdata</name>
  <symbols>
    <symbol>BEAS</symbol>
    <symbol>IBM</symbol>
  </symbols>
</adapter>
```

D.91 threshold

Use this element to define the threshold above which Oracle CEP server logs a monitoring event.

This element is applicable only in an `average-latency` element in a diagnostic profile.

For more information, see "Monitoring the Throughput and Latency of a Stage or Path in the EPN" in the *Oracle Complex Event Processing Visualizer User's Guide*.

D.91.1 Child Elements

The `threshold` component configuration element supports the following child elements:

- `amount`
- `unit`

D.91.2 Attributes

The `threshold` component has no attributes.

D.91.3 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>
  </profile>
</diagnostic-profiles>
```

```

<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>

```

D.92 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 Complex Event Processing Visualizer User's Guide*.

D.92.1 Child Elements

The throughput component configuration element supports the following child elements:

- [name](#)
- [throughput-interval](#)
- [average-interval](#)
- [location](#)

D.92.2 Attributes

The throughput component has no attributes.

D.92.3 Example

The following example shows how to use the `throughput` element in the component configuration file:

```

<diagnostic-profiles>
  <name>myselfprofiles</name>
  <profile>
    <name>testProfile0</name>
    <enabled>true</enabled>
    <start-stage>MetricSubscriber</start-stage>
    <throughput>
      <throughput-interval>
        <amount>100000</amount>
        <unit>MICROSECONDS</unit>
      </throughput-interval>
      <average-interval>
        <amount>100000000</amount>
        <unit>NANOSECONDS</unit>
      </average-interval>
    </throughput>
  </profile>
</diagnostic-profiles>

```

```
        <location>
          <application>diagnostic</application>
          <stage>AlertEventStream</stage>
          <direction>INBOUND</direction>
        </location>
      </throughput>
    </profile>
  </diagnostic-profiles>
```

D.93 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 Complex Event Processing Visualizer User's Guide*.

D.93.1 Child Elements

The `throughput-interval` component configuration element supports the following child elements:

- `amount`
- `unit`

D.93.2 Attributes

The `throughput-interval` component has no attributes.

D.93.3 Example

The following example shows how to use the `throughput-interval` element in the component configuration file:

```
<diagnostic-profiles>
  <name>myselfprofiles</name>
  <profile>
    <name>testProfile0</name>
    <enabled>true</enabled>
    <start-stage>MetricSubscriber</start-stage>
    <throughput>
      <throughput-interval>
        <amount>100000</amount>
        <unit>MICROSECONDS</unit>
      </throughput-interval>
      <average-interval>
        <amount>100000000</amount>
        <unit>NANOSECONDS</unit>
      </average-interval>
      <location>
        <application>diagnostic</application>
        <stage>AlertEventStream</stage>
        <direction>INBOUND</direction>
      </location>
    </throughput>
  </profile>
</diagnostic-profiles>
```

D.94 time-range

Use this element to define a filter that Oracle CEP server applies to the events in the event store. Only events with a record-time in this time range will be played back to the stage.

Use either `time-range-offset` or `time-range` but not both.

For more information, see [Chapter 13, "Configuring Event Record and Playback"](#).

D.94.1 Child Elements

The `time-range` component configuration element supports the following child elements:

- `start`
- `end`

D.94.2 Attributes

The `time-range` component has no attributes.

D.94.3 Example

The following example shows how to use the `time-range` element in the component configuration file:

```
<record-parameters>
  <dataset-name>tuple1</dataset-name>
  <event-type-list>
    <event-type>TupleEvent1</event-type>
  </event-type-list>
  <provider-name>test-rdbms-provider</provider-name>
  <store-policy-parameters>
    <parameter>
      <name>timeout</name>
      <value>300</value>
    </parameter>
  </store-policy-parameters>
  <time-range>
    <start>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>
```

D.95 time-range-offset

Use this element to define a filter that Oracle CEP server applies to the events in the event store. Only events with a record-time in this time range will be played back to the stage.

Use either `time-range` or `time-range-offset` but not both.

For more information, see [Chapter 13, "Configuring Event Record and Playback"](#).

D.95.1 Child Elements

The `time-range-offset` component configuration element supports the following child elements:

- `start`
- `duration`

D.95.2 Attributes

The `time-range-offset` component has no attributes.

D.95.3 Example

The following example shows how to use the `time-range-offset` element in the component configuration file:

```
<record-parameters>
  <dataset-name>tuple1</dataset-name>
  <event-type-list>
    <event-type>TupleEvent1</event-type>
  </event-type-list>
  <provider-name>test-rdbms-provider</provider-name>
  <store-policy-parameters>
    <parameter>
      <name>timeout</name>
      <value>300</value>
    </parameter>
  </store-policy-parameters>
  <time-range-offset>
    <start>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>
```

D.96 time-to-live

Use this element to define the maximum amount of time, in milliseconds, that an entry is cached. Default value is infinite.

For more information, see [Section 12.2, "Configuring an Oracle CEP Local Caching System and Cache"](#).

D.96.1 Child Elements

The `time-to-live` component configuration element has no child elements.

D.96.2 Attributes

The `time-to-live` component has no attributes.

D.96.3 Example

The following example shows how to use the `time-to-live` element in the component configuration file:

```
<caching-system>
  <name>providerCachingSystem</name>
  <cache>
    <name>providerCache</name>
    <max-size>1000</max-size>
    <eviction-policy>FIFO</eviction-policy>
    <time-to-live>60000</time-to-live>
    <idle-time>120000</idle-time>
  </cache>
</caching-system>
```



```

        <write-none/>
        <work-manager-name>JettyWorkManager</work-manager-name>
        <listeners asynchronous="false">
            <work-manager-name>JettyWorkManager</work-manager-name>
        </listeners>
    </cache>
</caching-system>

```

D.97 unit

Use this element to define the duration units of `amount` element.

Valid values are:

- NANoseconds
- MICROseconds
- MILLIseconds
- SECONDS
- MINUTES
- HOURS
- DAYS

D.97.1 Child Elements

The `unit` component configuration has no child elements:

D.97.2 Attributes

The `unit` component has no attributes.

D.97.3 Example

The following example shows how to use the `unit` element in the component configuration file:

```

<diagnostic-profiles>
  <name>myselfprofiles</name>
  <profile>
    <name>testProfile0</name>
    <enabled>true</enabled>
    <start-stage>MetricSubscriber</start-stage>
    <max-latency>
      <collect-interval>
        <amount>1000</amount>
        <unit>s</unit>
      </collect-interval>
      <name>testProfile0MaxLat</name>
      <start-location>
        <application>diagnostic</application>
        <stage>MetricSubscriber</stage>
        <direction>INBOUND</direction>
      </start-location>
      <end-location>
        <application>diagnostic</application>
        <stage>MonitorProcessor</stage>
        <direction>OUTBOUND</direction>
      </end-location>
    </max-latency>
  </profile>

```

```
</diagnostic-profiles>
```

D.98 user

Use the `user` element in the following parent elements:

- [http-pub-sub-adapter](#): Use the `user` element to define the user name if the HTTP pub-sub server to which the Oracle CEP application is publishing requires user authentication.
- [jms-adapter](#): When Oracle CEP acquires the JNDI `InitialContext`, it uses the `user` and [password](#) (or [encrypted-password](#)) elements. When Oracle CEP 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 CEP uses the `user` and [password](#) (or [encrypted-password](#)) elements.

D.98.1 Child Elements

The `user` component configuration element has no child elements.

D.98.2 Attributes

The `user` component has no attributes.

D.98.3 Example

The following example shows how to use the `user` element in the component configuration file:

```
<http-pub-sub-adapter>
  <name>remotePub</name>
  <server-url>http://localhost:9002/pubsub</server-url>
  <channel>/test1</channel>
  <event-type>com.bea.wlevs.tests.httppubsub.PubsubTestEvent</event-type>
  <user>wlevs</user>
  <password>wlevs</password>
</http-pub-sub-adapter>
```

D.99 value

Use this element to define the value of a [parameter](#) element.

D.99.1 Child Elements

The `value` component configuration element has no child elements.

D.99.2 Attributes

The `value` component has no attributes.

D.99.3 Example

The following example shows how to use the `value` element in the component configuration file:

```
<record-parameters>
  <dataset-name>tuple1</dataset-name>
```

```

<event-type-list>
  <event-type>TupleEvent1</event-type>
</event-type-list>
<provider-name>test-rdbms-provider</provider-name>
<store-policy-parameters>
  <parameter>
    <name>timeout</name>
    <value>300</value>
  </parameter>
</store-policy-parameters>
<batch-size>1</batch-size>
<batch-time-out>10</batch-time-out>
</record-parameters>

```

D.100 view

Use this element to define an Oracle CQL view for a component.

For more information, see [Chapter 10, "Configuring Oracle CQL Processors"](#).

D.100.1 Child Elements

The `view` component configuration element has no child elements.

D.100.2 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?
<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>schema</code>	Space delimited list of stream elements used in the view.	String of space delimited tokens.	No.

D.100.3 Example

The following example shows how to use the `view` element in the component configuration file:

```

<processor>
  <name>cqlProcessor</name>
  <rules>
    <view id="lastEvents" schema="cusip bid srcId bidQty ask askQty seq"><![CDATA[
      select cusip, bid, srcId, bidQty, ask, askQty, seq
      from inputChannel[partition by srcId, cusip rows 1]
    ]]></view>
    <view id="bidask" schema="cusip bid ask"><![CDATA[
      select cusip, max(bid), min(ask)
      from lastEvents
      group by cusip
    ]]></view>
    <view ...><![CDATA[
      ...
    ]]></view>
    ...
  </rules>
</processor>

```

```
<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>
```

D.101 work-manager

Use this element to define the name of a work manager for a [jms-adapter](#).

Valid value is the name specified in the Oracle CEP server `config.xml` file `work-manager` element name child element. The default value is the work manager configured for the application itself.

For more information, see [Section F.44, "work-manager"](#).

D.101.1 Child Elements

The `work-manager` component configuration element has no child elements:

D.101.2 Attributes

The `work-manager` component has no attributes.

D.101.3 Example

The following example shows how to use the `work-manager` element in the component configuration file:

```
<jms-adapter>
  <name>jmsInbound-text</name>
  <jndi-provider-url>t3://localhost:7001</jndi-provider-url>
  <destination-name>JMSServer-0/Module1!Queue1</destination-name>
  <user>weblogic</user>
  <password>weblogic</password>
  <work-manager>JettyWorkManager</work-manager>
  <concurrent-consumers>1</concurrent-consumers>
  <session-transacted>>false</session-transacted>
</jms-adapter>
```

D.102 work-manager-name

Use this element to define a work manager for a [cache](#).

The [listeners](#) element has a single child element, `work-manager-name`, that specifies the work manager to be used for asynchronously invoking listeners. This value is ignored if synchronous invocations are enabled. If a work manager is specified for the cache itself, this value overrides it for invoking listeners only.

Valid value is the name specified in the Oracle CEP server `config.xml` file `work-manager` element name child element. The default value is the work manager configured for the application itself.

D.102.1 Child Elements

The `work-manager-name` component configuration element has no child elements:

D.102.2 Attributes

The `work-manager-name` component has no attributes.

D.102.3 Example

The following example shows how to use the `work-manager-name` element in the component configuration file:

```
<cache>
  <name>providerCache</name>
  <max-size>1000</max-size>
  <eviction-policy>FIFO</eviction-policy>
  <time-to-live>60000</time-to-live>
  <idle-time>120000</idle-time>
  <write-none/>
  <work-manager-name>JettyWorkManager</work-manager-name>
  <listeners asynchronous="false">
    <work-manager-name>JettyWorkManager</work-manager-name>
  </listeners>
</cache>
```

D.103 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.

D.103.1 Child Elements

The `write-behind` component configuration element supports the following child elements:

- [work-manager-name](#)
- [batch-size](#)
- [buffer-size](#)
- [buffer-write-attempts](#)
- [buffer-write-timeout](#)

D.103.2 Attributes

The `write-behind` component has no attributes.

D.103.3 Example

The following example shows how to use the `write-behind` element in the component configuration file:

```
<caching-system>
```

```
<name>providerCachingSystem</name>
<cache>
  <name>providerCache</name>
  <max-size>1000</max-size>
  <eviction-policy>FIFO</eviction-policy>
  <time-to-live>60000</time-to-live>
  <idle-time>120000</idle-time>
  <write-behind>
    <work-manager-name>JettyWorkManager</work-manager-name>
    <batch-size>100</batch-size>
    <buffer-size>100</buffer-size>
    <buffer-write-attempts>100</buffer-write-attempts>
    <buffer-write-timeout>100</buffer-write-timeout>
  </write-behind>
  <work-manager-name>JettyWorkManager</work-manager-name>
  <listeners asynchronous="false">
    <work-manager-name>JettyWorkManager</work-manager-name>
  </listeners>
</cache>
</caching-system>
```

D.104 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.

D.104.1 Child Elements

The `write-none` component configuration element has no child elements.

D.104.2 Attributes

The `write-none` component has no attributes.

D.104.3 Example

The following example shows how to use the `write-none` element in the component configuration file:

```
<caching-system>
  <name>providerCachingSystem</name>
  <cache>
    <name>providerCache</name>
    <max-size>1000</max-size>
    <eviction-policy>FIFO</eviction-policy>
    <time-to-live>60000</time-to-live>
    <idle-time>120000</idle-time>
    <write-none/>
    <work-manager-name>JettyWorkManager</work-manager-name>
    <listeners asynchronous="false">
      <work-manager-name>JettyWorkManager</work-manager-name>
    </listeners>
  </cache>
</caching-system>
```

D.105 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.

D.105.1 Child Elements

The `write-through` component configuration element has no child elements.

D.105.2 Attributes

The `write-through` component has no attributes.

D.105.3 Example

The following example shows how to use the `write-through` element in the component configuration file:

```
< caching-system >
  < name>providerCachingSystem</ name >
  < cache >
    < name>providerCache</ name >
    < max-size>1000</ max-size >
    < eviction-policy>FIFO</ eviction-policy >
    < time-to-live>60000</ time-to-live >
    < idle-time>120000</ idle-time >
    < write-through / >
    < work-manager-name>JettyWorkManager</ work-manager-name >
    < listeners asynchronous="false" >
      < work-manager-name>JettyWorkManager</ work-manager-name >
    </ listeners >
  </ cache >
</ caching-system >
```

Schema Reference: Deployment deployment.xsd

This appendix describes the elements of the `deployment.xsd` schema.

For more information, see:

- [Section E.1, "Overview of the Oracle CEP Deployment Elements"](#)
- [Section B.3, "Deployment Schema deployment.xsd"](#)

E.1 Overview of the Oracle CEP Deployment Elements

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

E.1.1 Element Hierarchy

The Oracle CEP component configuration elements are organized into the following hierarchy:

beans

Standard Spring and OSGi elements such as `bean`, `osgi-service`, and so on.

E.1.2 Example of an Oracle CEP Deployment Configuration File

The following sample deployment configuration file from the `fx` application shows how to use many of the Oracle CEP elements:

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns:wlevs="http://www.bea.com/ns/wlevs/deployment" xsi:schemaLocation="
  http://www.springframework.org/schema/beans
  http://www.springframework.org/schema/beans/spring-beans.xsd
  http://www.bea.com/ns/wlevs/deployment
  http://www.bea.com/ns/wlevs/deployment/deployment.xsd">
  <bean class="org.springframework.beans.factory.config.PropertyPlaceholderConfigurer">
<property name="systemPropertiesModeName" value="SYSTEM_PROPERTIES_MODE_OVERRIDE"/>
</bean>
<wlevs:deployment
  id="fx"
  state="start"
  location="file:${wlevs.domain.home}/applications/fx/com.bea.wlevs.example.fx_11.1.0.0.jar"/>
</beans>
```

E.2 wlevs:deployment

Use this element to declare an adapter component to the Spring application context.

E.2.1 Child Elements

The `wlevs:deployment` deployment element has no child elements:

E.2.2 Attributes

[Table E-1](#) lists the attributes of the `wlevs:deployment` deployment element.

Table E-1 Attributes of the `wlevs:deployment` Deployment Element

Attribute	Description	Data Type	Required?
<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 CEP server.	String	No.
<code>state</code>	Specifies the state that the bundle should be in once it is deployed to the Oracle CEP server. The value of this attribute must be one of the following: <ul style="list-style-type: none"> ▪ <code>start</code>: Install and start the bundle so that it immediately begins taking client requests. ▪ <code>install</code>: Install the bundle, but do not start it. ▪ <code>update</code>: Update an existing bundle. Default value: <code>start</code> .	String	No.

E.2.3 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 describes the elements of the `wlevs_server_config.xsd` schema.

For more information, see:

- [Section F.1, "Overview of the Oracle CEP Server Configuration Elements"](#)
- [Section B.4, "Server Configuration Schema `wlevs_server_config.xsd`"](#)

F.1 Overview of the Oracle CEP Server Configuration Elements

Oracle CEP provides a number of server configuration elements that you use to configure Oracle CEP server-specific attributes and services.

F.1.1 Element Hierarchy

The top-level Oracle CEP server configuration elements are organized into the following hierarchy:

- `config`
 - `domain`
 - `rmi`
 - `jndi-context`
 - `exported-jndi-context`
 - `jmx`
 - `transaction-manager`
 - `work-manager`
 - `logging-service`
 - `log-stdout`
 - `log-file`
 - `jetty-web-app`
 - `netio`
 - `jetty`
 - `netio-client`
 - `debug`

- [data-source](#)
- [http-pubsub](#)
- [event-store](#)
- [cluster](#)
- [bdb-config](#)
- [rdbms-event-store-provider](#)
- [ssl](#)
- [weblogic-rmi-client](#)
- [weblogic-jta-gateway](#)
- [use-secure-connections](#)
- [show-detail-error-message](#)
- [cql](#)

F.1.2 Example of an Oracle CEP Server Configuration File

The following sample Oracle CEP server configuration file from the HelloWorld application shows how to use many of the Oracle CEP elements:

```
<?xml version="1.0" encoding="UTF-8"?>
<n1:config xsi:schemaLocation="http://www.bea.com/ns/wlevs/config/server wlevs_server_config.xsd"
  xmlns:n1="http://www.bea.com/ns/wlevs/config/server"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <domain>
    <name>WLEventServerDomain</name>
  </domain>

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

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

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

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

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

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

```

</jndi-context>

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

<jmx>
  <rmi-service-name>RMI</rmi-service-name>
  <rmi-jrmp-port>9999</rmi-jrmp-port>
  <jndi-service-name>JNDI</jndi-service-name>
  <rmi-registry-port>9004</rmi-registry-port>
</jmx>

<ssl>
  <name>sslConfig</name>
  <key-store>./ssl/evsidentity.jks</key-store>
  <key-store-pass>
    <password>{Salted-3DES}s4YUEvH4Wl2DAjb45iJnrw==</password>
  </key-store-pass>
  <key-store-alias>evsidentity</key-store-alias>
  <key-manager-algorithm>SunX509</key-manager-algorithm>
  <ssl-protocol>TLS</ssl-protocol>
  <enforce-fips>false</enforce-fips>
  <need-client-auth>false</need-client-auth>
</ssl>

<http-pubsub>
  <name>pubsub</name>
  <path>/pubsub</path>
  <pub-sub-bean>
    <server-config>
      <supported-transport>
        <types>
          <element>long-polling</element>
        </types>
      </supported-transport>
      <publish-without-connect-allowed>true</publish-without-connect-allowed>
    </server-config>
    <channels>
      <element>
        <channel-pattern>/evsmonitor</channel-pattern>
      </element>
      <element>
        <channel-pattern>/evsalert</channel-pattern>
      </element>
      <element>
        <channel-pattern>/evsdomainchange</channel-pattern>
      </element>
    </channels>
  </pub-sub-bean>
</http-pubsub>

<!-- Sample cluster configuration -->
<!--
<cluster>
  <server-name>myServer</server-name>
  <multicast-address>239.255.0.1</multicast-address>
  <enabled>coherence</enabled>
  <security>none</security>
  <groups></groups>
</cluster>
-->

<logging-service>
  <name>myLogService</name>

```

```

<log-file-config>myFileConfig</log-file-config>
<stdout-config>myStdoutConfig</stdout-config>
<logger-severity>Notice</logger-severity>
<!-- logger-severity-properties is used to selectively enable logging for
individual categories -->
<!--logger-severity-properties>
  <entry>
    <key>org.springframework.osgi.extender.internal.dependencies.startup</key>
    <value>Debug</value>
  </entry>
</logger-severity-properties-->
</logging-service>

<log-file>
  <name>myFileConfig</name>
  <rotation-type>none</rotation-type>
</log-file>

<log-stdout>
  <name>myStdoutConfig</name>
  <stdout-severity>Debug</stdout-severity>
</log-stdout>

</nl:config>

```

F.2 auth-constraint

Use this element to configure an authorization constraint for a [channel-constraints](#) element.

For more information on channels, see [channels](#).

F.2.1 Child Elements

The `auth-constraint` server configuration element supports the child elements that [Table F-1](#) lists

Table F-1 Child Elements of: `auth-constraint`

XML Tag	Type	Description
<code>description</code>	string	The description of the role.
<code>role-name</code>	string	A valid role name. "Users, Groups, and Roles" in the <i>Oracle Complex Event Processing Administrator's Guide</i>

F.2.2 Attributes

The `auth-constraint` server configuration element has no attributes.

F.2.3 Example

The following example shows how to use the `auth-constraint` element in the Oracle CEP server configuration file:

```

<http-pubsub>
  <name>myPubsub</name>
  <path>/pubsub</path>
  <pub-sub-bean>
    ...
    <channel-constraints>

```

```

        <element>
...
            <auth-constraint>
                <description>Administrators</description>
                <role-name>admin</role-name>
            </auth-constraint>
        </element>
    </channel-constraints>
</pub-sub-bean>
</http-pubsub>

```

F.3 bdb-config

Use this element to configure the default event store provider that uses a Berkeley database instance.

Optionally, you may configure Oracle CEP server to use a relational database instance as the event store provider as [Section F.31, "rdbms-event-store-provider"](#) describes.

F.3.1 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
<code>db-env-path</code>	string	Specifies the subdirectory in which Oracle CEP server creates Berkeley database instances relative to the 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>
<code>cache-size</code>	long	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"> ▪ http://www.oracle.com/technology/documentation/berkeley-db/je/GettingStartedGuide/cachesize.html. ▪ http://www.oracle.com/technology/documentation/berkeley-db/je/java/com/sleepycat/je/EnvironmentMutableConfig.html#setCacheSize(long) Default: <code>je.maxMemoryPercent * JVM maximum memory</code>

F.3.2 Attributes

The `bdb-config` server configuration element has no attributes.

F.3.3 Example

The following example shows how to use the `bdb-config` element in the Oracle CEP server configuration file:

```

<bdb-config>
  <db-env-path>bdb</db-env-path>
  <cache-size>1000</cache-size>

```

```
</bdb-config>
```

F.4 channels

Use this element to configure one or more channels for a [pubsub-bean](#) element.

Channel patterns always begin with a forward slash (/). Clients subscribe to these channels to either publish or receive messages

F.4.1 Child Elements

The `channels` server configuration element contains one or more `element` child elements that each contain a `channel-pattern` child element and zero or more `message-filters` child elements. Each `message-filters` child element contains an `element` child element with the string value of a `message-filter-name` that corresponds to a [message-filters](#) element.

F.4.2 Attributes

The `channels` server configuration element has no attributes.

F.4.3 Example

The following example shows how to use the `channels` element in the Oracle CEP server configuration file:

```
<http-pubsub>
  <name>myPubsub</name>
  <path>/pubsub</path>
  <pub-sub-bean>
    <server-config>
      <supported-transport>
        <types>
          <element>long-polling</element>
        </types>
      </supported-transport>
      <publish-without-connect-allowed>
        true
      </publish-without-connect-allowed>
    </server-config>
    <channels>
      <element>
        <channel-pattern>/evsmonitor</channel-pattern>
      </element>
      <element>
        <channel-pattern>/evsalert</channel-pattern>
      </element>
      <element>
        <channel-pattern>/evsdomainchange</channel-pattern>
      </element>
    </channels>
  </pub-sub-bean>
</http-pubsub>
```


F.5 channel-constraints

Use this element to configure one or more channel constraints for a [pubsub-bean](#) element.

For more information on channels, see [channels](#).

F.5.1 Child Elements

The `channel-constraints` server configuration element contains one or more `element` child element that each support the following child elements:

- [channel-resource-collection](#)
- [auth-constraint](#)

F.5.2 Attributes

The `channel-constraints` server configuration element has no attributes.

F.5.3 Example

The following example shows how to use the `channel-constraints` element in the Oracle CEP server configuration file:

```
<http-pubsub>
  <name>myPubsub</name>
  <path>/pubsub</path>
  <pub-sub-bean>
  ...
    <channel-constraints>
      <element>
        <channel-resource-collection>
          <element>
            <channel-resource-name>Foo</channel-resource-name>
            <descriptions>
              <element>Foo</element>
            </descriptions>
            <channel-patterns>
              <element>Foo</element>
            </channel-patterns>
            <channel-operations>
              <element>Foo</element>
            </channel-operations>
          </element>
        </channel-resource-collection>
        <auth-constraint>
          <description>Foo</description>
          <role-name>Foo</role-name>
        </auth-constraint>
      </element>
    </channel-constraints>
  </pub-sub-bean>
</http-pubsub>
```

F.6 channel-resource-collection

Use this element to configure one or more channel resource collections for a [channel-constraints](#) element.

For more information on channels, see [channels](#).

F.6.1 Child Elements

The `channel-resource-collection` server configuration element contains zero or more `element` child elements that support the child elements that [Table F-3](#) lists

Table F-3 *Child Elements of: channel-resource-collection*

XML Tag	Type	Description
<code>channel-resource-name</code>	string	The name of this channel resource.
<code>descriptions</code>	string	Description of this channel resource collection. This element contains an <code>element</code> child element with a string value.
<code>channel-patterns</code>	string	Specifies a channel pattern. This element contains an <code>element</code> child element with a string value.
<code>channel-operations</code>	string	Specifies the operation to channel, validate values include: <ul style="list-style-type: none"> ▪ create ▪ delete ▪ subscribe ▪ publish This element contains an <code>element</code> child element with a string value.

F.6.2 Attributes

The `channel-resource-collection` server configuration element has no attributes.

F.6.3 Example

The following example shows how to use the `channel-resource-collection` element in the Oracle CEP server configuration file:

```
<http-pubsub>
  <name>myPubsub</name>
  <path>/pubsub</path>
  <pub-sub-bean>
  ...
    <channel-constraints>
      <element>
        <channel-resource-collection>
          <element>
            <channel-resource-name>Foo</channel-resource-name>
            <descriptions>
              <element>Foo</element>
            </descriptions>
            <channel-patterns>
              <element>Foo</element>
            </channel-patterns>
            <channel-operations>
              <element>Foo</element>
            </channel-operations>
          </element>
        </channel-resource-collection>
      <auth-constraint>
        <description>Foo</description>
```

```

        <role-name>Foo</role-name>
    </auth-constraint>
</element>
</channel-constraints>
</pub-sub-bean>
</http-pubsub>

```

F.7 cluster

Use this element to configure a cluster component in the Oracle CEP server.

For more information, see "Administrating Multi-Server Domains With Oracle CEP Native Clustering" in the *Oracle Complex Event Processing Administrator's Guide*.

F.7.1 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 name .
<code>server-name</code>	string	Specifies a unique name for the server. Oracle CEP Visualizer uses the value of this element when it displays the server in its console. Default value: <ul style="list-style-type: none"> Oracle CEP native clustering: <code>WLEvServer-identity</code> where <i>identity</i> is the value of the <code>identity</code> element. Oracle Coherence: <code>WLEvServer-identity</code> where <i>identity</i> is the member ID as determined by Oracle Coherence.
<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.
<code>multicast-address</code>	string	This child element is required unless all servers of the multi-server domain are hosted on the same computer; in that case you can omit the <code>multicast-address</code> element and Oracle CEP automatically assigns a multicast address to the multi-server domain based on the computer's IP address. If, however, the servers are hosted on different computers, then you must provide an appropriate domain-local address. Oracle recommends you use an address of the form <code>239.255.X.X</code> , which is what the auto-assigned multicast address is based on. All the Oracle CEP servers using this <code>multicast-address</code> must be on the same subnet. 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.
<code>multicast-interface</code>	string	The name of the interface that the multicast address should be bound to. This can be one of: <ul style="list-style-type: none"> Simple name, such as <code>eth0</code>. IP address to which the NIC is bound, such as <code>192.168.1.2</code>. 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>.

Table F–4 (Cont.) Child Elements of: cluster

XML Tag	Type	Description
multicast-port	int	Specifies the port used for multicast traffic. Default value is 9100.
identity	string	Applicable only to Oracle CEP native clustering; specifies the server's identity and must be an integer between 1 and INT_MAX. Oracle CEP numerically compares the server identities during multi-server operations; the server with the lowest identity becomes the domain coordinator. Be sure that each server in the multi-server domain has a different identity; if servers have the same identity, the results of multi-server operations are unpredictable. Not applicable to Oracle Coherence.
enabled	See Description	Specifies whether or not the cluster is enabled. Valid values: <ul style="list-style-type: none"> ▪ coherence ▪ evs4j ▪ true: cluster is enabled (Oracle Coherence mode) ▪ false: cluster is not enabled (default).
security	See Description	Specifies the type of security for this cluster. Valid values: <ul style="list-style-type: none"> ▪ none—Default value. Specifies that no security is configured for the multi-server domain. ▪ encrypt—Specifies that multi-server messages should be encrypted.
groups	string	Specifies a comma-separated list of the names of the groups this cluster belongs to. For more information, see "Groups" in the <i>Oracle Complex Event Processing Administrator's Guide</i> .
operation-timeout	int	Specifies, in milliseconds, the timeout for point-to-point HTTP multi-server requests. Default value is 30000.

F.7.2 Attributes

The `cluster` server configuration element has no attributes.

F.7.3 Example

The following example shows how to use the `cluster` element in the Oracle CEP server configuration file:

```
<cluster>
  <name>MyCluster</name>
  <server-name>myServer1</server-name>
  <multicast-address>239.255.0.1</multicast-address>
  <identity>1</identity>
  <enabled>true</enabled>
</cluster>
```

In the example, the `cluster` element's unique identifier is `MyCluster`.

F.8 connection-pool-params

Use this element to specify connection pool-related [data-source](#) parameters.

F.8.1 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
statement-timeout	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.
profile-harvest-frequency-seconds	int	The number of seconds between diagnostic profile harvest operations. Default: 300.
inactive-connection-timeout-seconds	int	The number of inactive seconds on a reserved connection before the connection is reclaimed and released back into the connection pool. Default: 0.
shrink-frequency-seconds	int	The number of seconds to wait before shrinking a connection pool that has incrementally increased to meet demand. Default: 900.
driver-interceptor	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.
seconds-to-trust-an-idle-pool-connection	int	The number of seconds within a connection use that the server trusts that the connection is still viable and will skip the connection test, either before delivering it to an application or during the periodic connection testing process. Default: 10.
pinned-to-thread	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> .
test-connections-on-reserve	boolean	Test a connection before giving it to a client. Requires that you specify <code>test-table-name</code> . Default: <code>false</code> .
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"> ■ LRU - when a new prepared or callable statement is used, the least recently used statement is replaced in the cache ■ FIXED - the first fixed number of prepared and callable statements are cached Default: LRU.
connection-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: <code>false</code> .
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.

Table F-5 (Cont.) Child Elements of: connection-pool-params

XML Tag	Type	Description
test-table-name	string	The name of the database table to use when testing physical database connections. This name is required when you specify test-frequency-seconds and enable test-reserved-connections. The default SQL code used to test a connection is <code>select count(*) from test-table-name</code> where <code>test-table-name</code> is the value of the test-table-name element. Most database servers optimize this SQL to avoid a table scan, but it is still a good idea to set test-table-name to the name of a table that is known to have few rows, or even no rows. If test-table-name begins with <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.
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-num-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.

F.8.2 Attributes

The `connection-pool-params` server configuration element has no attributes.

F.8.3 Example

The following example shows how to use the `connection-pool-params` element in the Oracle CEP server configuration file:

```

<data-source>
  <name>orads</name>
  <xa-params>
    <keep-xa-conn-till-tx-complete>true</keep-xa-conn-till-tx-complete>
  </xa-params>
  <driver-params>
    <url>jdbc:oracle:thin:@localhost:1521:ce102</url>
    <driver-name>oracle.jdbc.OracleDriver</driver-name>
    <properties>
      <element>
        <name>user</name>
        <value>wlevs</value>
      </element>
      <element>
        <name>password</name>
        <value>wlevs</value>
      </element>
    </properties>
  </driver-params>
  <connection-pool-params>
    <initial-capacity>5</initial-capacity>
    <max-capacity>10</max-capacity>
    <test-table-name>SQL SELECT 1 FROM DUAL</test-table-name>
    <test-frequency-seconds>5</test-frequency-seconds>
  </connection-pool-params>
  <data-source-params>
    <jndi-names>
      <element>orads</element>
    </jndi-names>
    <global-transactions-protocol>None</global-transactions-protocol>
  </data-source-params>
</data-source>

```

F.9 cql

Use this element to configure Oracle CQL-specific options in the Oracle CEP server.

F.9.1 Child Elements

The `cql` server configuration element supports the following child elements:

- [name](#)
- [scheduler](#)

F.9.2 Attributes

The `cql` server configuration element has no attributes.

F.9.3 Example

The following example shows how to use the `cql` element in the Oracle CEP server configuration file:

```

<cql>
  <name>myCQL</name>
  <storage>
    <folder>myfolder</folder>
    <metadata-name>myname</metadata-name>
  </storage>
  <scheduler>
    <class-name>myclass</class-name>
  </scheduler>
</cql>

```

```
        <threads>10</threads>
        <direct-interop>>false</direct-interop>
    </scheduler>
</cql>
```

In the example, the `cql` element's unique identifier is `myCQL`.

F.10 data-source

This configuration type defines configuration for a DataSource service.

F.10.1 Child Elements

The `data-source` server configuration element supports the following child elements:

- `name`
- `xa-params`
- `data-source-params`
- `connection-pool-params`
- `driver-params`

F.10.2 Attributes

The `data-source` server configuration element has no attributes.

F.10.3 Example

The following example shows how to use the `data-source` element in the Oracle CEP server configuration file:

```
<data-source>
  <name>orads</name>
  <driver-params>
    <url>jdbc:oracle:thin:@localhost:1521:ce102</url>
    <driver-name>oracle.jdbc.OracleDriver</driver-name>
    <properties>
      <element>
        <name>user</name>
        <value>wlevs</value>
      </element>
      <element>
        <name>password</name>
        <value>wlevs</value>
      </element>
    </properties>
  </driver-params>
  <connection-pool-params>
    <initial-capacity>5</initial-capacity>
    <max-capacity>10</max-capacity>
    <test-table-name>SQL SELECT 1 FROM DUAL</test-table-name>
    <test-frequency-seconds>5</test-frequency-seconds>
  </connection-pool-params>
  <data-source-params>
    <jndi-names>
      <element>orads</element>
    </jndi-names>
    <global-transactions-protocol>None</global-transactions-protocol>
  </data-source-params>
```



```
</data-source>
```

In the example, the `data-source` element's unique identifier is `orads`.

F.11 data-source-params

Use this element to specify data source-related `data-source` parameters.

F.11.1 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"> Failover Load-Balancing 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.
<code>data-source-list</code>	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.
<code>failover-request-if-busy</code>	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.
<code>row-prefetch-size</code>	int	If row prefetching is enabled, specifies the number of result set rows to prefetch for a client between 2 and 65536. Default: 48.
<code>jndi-names</code>	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"> <code>element</code>: contains the string name of a valid data-source element. For more information, see data-source. <code>config-data-source-DataSourceParams-JNDINames</code>.
<code>scope</code>	boolean	Specifies the scoping of the data source. Note that <code>Global</code> is the only scoped supported by MSA. Default: <code>Global</code> .
<code>connection-pool-failover-callback-handler</code>	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.

Table F-6 (Cont.) Child Elements of: data-source-params

XML Tag	Type	Description
global-transactions-protocol	int	<p>Determines the transaction protocol (global transaction processing behavior) for the data source. Valid values:</p> <ul style="list-style-type: none"> ▪ TwoPhaseCommit - Standard XA transaction processing. Requires an XA driver ▪ LoggingLastResource - A performance enhancement for one non-XA resource ▪ EmulateTwoPhaseCommit - Enables one non-XA resource to participate in a global transaction, but has some risk to data ▪ OnePhaseCommit - One-phase XA transaction processing using a non-XA driver. This is the default setting ▪ None - Support for local transactions only <p>Default: OnePhaseCommit.</p>

F.11.2 Attributes

The data-source-params server configuration element has no attributes.

F.11.3 Example

The following example shows how to use the data-source-params element in the Oracle CEP server configuration file:

```

<data-source>
  <name>orads</name>
  <xa-params>
    <keep-xa-conn-till-tx-complete>>true</keep-xa-conn-till-tx-complete>
  </xa-params>
  <driver-params>
    <url>jdbc:oracle:thin:@localhost:1521:ce102</url>
    <driver-name>oracle.jdbc.OracleDriver</driver-name>
    <properties>
      <element>
        <name>user</name>
        <value>wlevs</value>
      </element>
      <element>
        <name>password</name>
        <value>wlevs</value>
      </element>
    </properties>
  </driver-params>
  <connection-pool-params>
    <initial-capacity>5</initial-capacity>
    <max-capacity>10</max-capacity>
    <test-table-name>SQL SELECT 1 FROM DUAL</test-table-name>
    <test-frequency-seconds>5</test-frequency-seconds>
  </connection-pool-params>
  <data-source-params>
    <jndi-names>
      <element>orads</element>
    </jndi-names>
    <global-transactions-protocol>None</global-transactions-protocol>
  </data-source-params>
</data-source>

```

F.12 driver-params

Use this element to specify JDBC driver-related [data-source](#) parameters.

F.12.1 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: <code>true</code> .
<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 element child elements that contain child elements: <ul style="list-style-type: none"> ▪ <code>name</code>: the property name. ▪ <code>value</code>: the property value.

F.12.2 Attributes

The `driver-params` server configuration element has no attributes.

F.12.3 Example

The following example shows how to use the `driver-params` element in the Oracle CEP server configuration file:

```
<data-source>
  <name>orads</name>
  <xa-params>
    <keep-xa-conn-till-tx-complete>true</keep-xa-conn-till-tx-complete>
  </xa-params>
  <driver-params>
    <url>jdbc:oracle:thin:@localhost:1521:ce102</url>
    <driver-name>oracle.jdbc.OracleDriver</driver-name>
    <properties>
      <element>
        <name>user</name>
        <value>wlevs</value>
      </element>
      <element>
        <name>password</name>
        <value>wlevs</value>
      </element>
    </properties>
  </driver-params>
  <connection-pool-params>
    <initial-capacity>5</initial-capacity>
    <max-capacity>10</max-capacity>
```

```

        <test-table-name>SQL SELECT 1 FROM DUAL</test-table-name>
        <test-frequency-seconds>5</test-frequency-seconds>
    </connection-pool-params>
    <data-source-params>
        <jndi-names>
            <element>orads</element>
        </jndi-names>
        <global-transactions-protocol>None</global-transactions-protocol>
    </data-source-params>
</data-source>

```

F.13 domain

Use this element to configure a domain name in the Oracle CEP server.

F.13.1 Child Elements

The `domain` server configuration element supports the following child elements:

- [name](#)

F.13.2 Attributes

The `domain` server configuration element has no attributes.

F.13.3 Example

The following example shows how to use the `domain` element in the Oracle CEP server configuration file:

```

<domain>
    <name>WLEventServerDomain</name>
</domain>

```

In the example, the domain's unique identifier is `WLEventServerDomain`.

F.14 debug

Use this element to configure one or more debug properties for the Oracle CEP server.

F.14.1 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
<code>name</code>	string	The name of this debug configuration. For more information, see name .
<code>debug-properties</code>	string	One or more child elements formed by taking a debug flag name (without its package name) and specifying a value of <code>true</code> . For more information including a full list of all debug flags, see "How to Configure Oracle CEP Debugging Options Using a Configuration File" in the <i>Oracle Complex Event Processing Administrator's Guide</i>

F.14.2 Attributes

The `debug` server configuration element has no attributes.

F.14.3 Example

The following example shows how to use the `debug` element to turn on Simple Declarative Services (SDS) debugging using debug flag `com.bea.core.debug.DebugSDS` in the Oracle CEP server configuration file.

```
<debug>
  <name>myDebug</name>
  <debug-properties>
    <DebugSDS>true</DebugSDS>
  ...
  </debug-properties>
</debug>
```

F.15 event-store

Use this element to configure an event store for the Oracle CEP server.

F.15.1 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
<code>name</code>	string	The name of this debug configuration. For more information, see name .
<code>provider-order</code>	string	Specifies the name of one or more <code>provider</code> child elements in the order in which the Oracle CEP server should access them. For more information, see: <ul style="list-style-type: none"> rdbms-event-store-provider

F.15.2 Attributes

The `event-store` server configuration element has no attributes.

F.15.3 Example

The following example shows how to use the `event-store` element in the Oracle CEP server configuration file:

```
<config>
  <event-store>
    <name>myEventStore</name>
    <provider-order>
      <provider>provider1</provider>
      <provider>provider2</provider>
    </provider-order>
  </event-store>
</config>
```

In the example, the adapter's unique identifier is `myEventStore`.

F.16 exported-jndi-context

This configuration type is used to export a remote JNDI service that may be accessed via clients using RMI. It registers the JNDI context with the RMI service, so that it may be accessed remotely by clients that pass a provider URL parameter when they create their `InitialContext` object. This service requires that a `jndi-context` configuration object also be specified. If it is not, then this service will not be able to start.

F.16.1 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
<code>name</code>	string	The name of this debug configuration. For more information, see name .
<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 rmi .

F.16.2 Attributes

The `exported-jndi-context` server configuration element has no attributes.

F.16.3 Example

The following example shows how to use the `exported-jndi-context` element in the Oracle CEP server configuration file:

```
<rmi>
  <name>myRMI</name>
  <http-service-name>TestJetty</http-service-name>
</rmi>

<exported-jndi-context>
  <name>RemoteJNDI</name>
  <rmi-service-name>myRMI</rmi-service-name>
</exported-jndi-context>
```

In the example, the adapter's unique identifier is `RemoteJNDI`.

F.17 http-pubsub

Use this element to configure an HTTP publish-subscribe service.

F.17.1 Child Elements

The `http-pubsub` server configuration element supports the following child elements:

- [name](#)
- [path](#)
- [pubsub-bean](#)

F.17.2 Attributes

The `http-pubsub` server configuration element has no attributes.

F.17.3 Example

The following example shows how to use the `http-pubsub` element in the Oracle CEP server configuration file:

```
<http-pubsub>
  <name>myPubsub</name>
  <path>/pubsub</path>
  <pub-sub-bean>
    <server-config>
      <supported-transport>
        <types>
          <element>long-polling</element>
        </types>
      </supported-transport>
      <publish-without-connect-allowed>
        true
      </publish-without-connect-allowed>
    </server-config>
    <channels>
      <element>
        <channel-pattern>/evsmonitor</channel-pattern>
      </element>
      <element>
        <channel-pattern>/evsalert</channel-pattern>
      </element>
      <element>
        <channel-pattern>/evsdomainchange</channel-pattern>
      </element>
    </channels>
  </pub-sub-bean>
</http-pubsub>
```

In the example, the `http-pubsub` element's unique identifier is `myPubsub`.

F.18 jetty

Use this element to configure an instance of the Jetty HTTP server.

F.18.1 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 name .
<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.

Table F–11 (Cont.) Child Elements of: jetty

XML Tag	Type	Description
work-manager-name	string	The name of the Work Manager that should be used for thread pooling. If this parameter is not specified, then Jetty will use a default work manager. For more information, see work-manager .
scratch-directory	string	The name of a directory where temporary files required for Web applications, JSPs, and other types of Web artifacts are kept. This parameter is overridden by the <code>scratch-directory</code> parameter on the <code>jetty-web-app</code> element. If this directory does not exist, it will be created.
debug-enabled	boolean	Enable debugging in the Jetty code. Specified debug messages are logged the same way as all other Debug level messages in the log service.
listen-port	int	The name of the network port that should be set. This parameter may not be set if the <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.
secure-network-io-name	string	The name of the Network I/O service that should be used for secure communications. The specified service must be configured to support SSL encryption. This parameter must refer to the name of a valid <code>netio</code> configuration object.

F.18.2 Attributes

The `jetty` server configuration element has no attributes.

F.18.3 Example

The following example shows how to use the `jetty` element in the Oracle CEP server configuration file:

```
<jetty>
  <name>TestJetty</name>
  <work-manager-name>WM</work-manager-name>
  <network-io-name>Netio</network-io-name>
  <secure-network-io-name>SecureNetio</secure-network-io-name>
  <debug-enabled>>false</debug-enabled>
  <scratch-directory>JettyWork</scratch-directory>
</jetty>
```

In the example, the `jetty` element's unique identifier is `TestJetty`.

F.19 jetty-web-app

Use this element to represent a Web application for use by Jetty. Each instance of this object represents a Web application which must be deployed using the Jetty service.

F.19.1 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
name	string	The name of this <code>jetty-web-app</code> element. For more information, see name .

Table F–12 (Cont.) Child Elements of: jetty-web-app

XML Tag	Type	Description
context-path	string	The context path where this web app will be deployed in the web server's name space. Default: /
scratch-directory	string	The location where Jetty should store temporary files for this web app. This parameter overrides the <code>scratch-directory</code> parameter on the <code>jetty</code> element. If this directory does not exist, it will be created.
path	string	A file name that points to the location of the web app on the server. It may be a directory or a WAR file.
jetty-name	string	The name of the Jetty service where this Web application should be deployed. This name must match the name of an existing <code>jetty</code> configuration object. For more information, see <code>jetty</code> .

F.19.2 Attributes

The `jetty-web-app` server configuration element has no attributes.

F.19.3 Example

The following example shows how to use the `jetty-web-app` element in the Oracle CEP server configuration file:

```
<jetty-web-app>
  <name>financial</name>
  <context-path>/financial</context-path>
  <path>../testws2/financialWS.war</path>
  <jetty-name>TestJetty</jetty-name>
</jetty-web-app>
```

In the example, the `jetty-web-app` element's unique identifier is `financial`.

F.20 jmx

Use this element to configure Java Management Extension (JMX) properties in the Oracle CEP server.

F.20.1 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
name	string	The name of this debug configuration. For more information, see name .
rmi-service-name	string	The name of the RMI service that should be used to serve this JNDI context over the network. It must match an existing RMI object in the configuration. For more information, see rmi .
jndi-service-name	string	The name of the JNDI service to which the JMX server will bind its object.

F.20.2 Attributes

The `jmx` server configuration element has no attributes.

F.20.3 Example

The following example shows how to use the `jmx` element in the Oracle CEP server configuration file:

```
<jmx>
  <name>myJMX</name>
  <jndi-service-name>JNDI</jndi-service-name>
  <rmi-service-name>RMI</rmi-service-name>
</jmx>
```

In the example, the `jmx` element's unique identifier is `myJMX`.

F.21 jndi-context

This configuration object is used to configure the JNDI provider. When it is placed in the configuration, the MSA JNDI Context is initialized. One instance of this configuration type must be placed in the configuration if the JNDI service is to be used, either locally, or remotely through the `exported-jndi-context` configuration type.

F.21.1 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 name .
<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.

F.21.2 Attributes

The `jndi-context` server configuration element has no attributes.

F.21.3 Example

The following example shows how to use the `jndi-context` element in the Oracle CEP server configuration file:

```
<jndi-context>
  <name>myJNDI</name>
  <default-provider>true</default-provider>
</jndi-context>
```

In the example, the adapter's unique identifier is `myJNDI`.

F.22 log-file

Use this element to configure logging to a file on the Oracle CEP server.

F.22.1 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 <code>work-manager</code> element. For more information, see name .
<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"> ▪ <code>bySize</code> ▪ <code>byTime</code> ▪ <code>none</code> 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> .
<code>log-file-severity</code>	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"> ▪ <code>Emergency</code> ▪ <code>Alert</code> ▪ <code>Critical</code> ▪ <code>Error</code> ▪ <code>Warning</code> ▪ <code>Notice</code> ▪ <code>Info</code> ▪ <code>Debug</code> ▪ <code>Trace</code> Default: <code>Notice</code> .
<code>log-file-rotation-dir</code>	string	The directory where the old rotated files are stored. If not set, the old files are stored in the same directory as the base log file.

F.22.2 Attributes

The `log-file` server configuration element has no attributes.

F.22.3 Example

The following example shows how to use the `log-file` element in the Oracle CEP server configuration file:

```
<log-file>
  <name>logFile</name>
  <number-of-files-limited>true</number-of-files-limited>
  <rotated-file-count>4</rotated-file-count>
  <rotate-log-on-startup-enabled>true</rotate-log-on-startup-enabled>
</log-file>
```

In the example, the `log-file` element's unique identifier is `logFile`.

F.23 log-stdout

Use this element to configure logging to standard out (console) on the Oracle CEP server.

F.23.1 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
<code>name</code>	string	The name of this <code>work-manager</code> element. For more information, see name .
<code>stack-trace-depth</code>	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> .
<code>stack-trace-enabled</code>	boolean	Specifies whether to dump stack traces to the console when included in a logged message. Default: <code>true</code> .
<code>stdout-severity</code>	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"> ▪ Emergency ▪ Alert ▪ Critical ▪ Error ▪ Warning ▪ Notice ▪ Info ▪ Debug ▪ Trace Default: <code>Notice</code> .

F.23.2 Attributes

The `log-stdout` server configuration element has no attributes.

F.23.3 Example

The following example shows how to use the `log-stdout` element in the Oracle CEP server configuration file:

```
<log-stdout>
  <name>logStdout</name>
  <stdout-severity>Debug</stdout-severity>
</log-stdout>
```

In the example, the `log-stdout` element's unique identifier is `logStdout`.

F.24 logging-service

Use this element to configure a logging service on the Oracle CEP server.

F.24.1 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
<code>name</code>	string	The name of this <code>work-manager</code> element. For more information, see name .
<code>log-file-config</code>	string	The configuration of the log file and its rotation policies.
<code>stdout-config</code>	string	The configuration of the stdout output.
<code>logger-severity</code>	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"> ▪ <code>Emergency</code> ▪ <code>Alert</code> ▪ <code>Critical</code> ▪ <code>Error</code> ▪ <code>Warning</code> ▪ <code>Notice</code> ▪ <code>Info</code> ▪ <code>Debug</code> ▪ <code>Trace</code> Default: <code>Info</code> .
<code>logger-severity-properties</code>	See Description	The Severity values for different nodes in the Logger tree composed of one or more <code>entry</code> child elements each containing a <code>key</code> and <code>value</code> child element.

F.24.2 Attributes

The `logging-service` server configuration element has no attributes.

F.24.3 Example

The following example shows how to use the `logging-service` element in the Oracle CEP server configuration file:

```
<logging-service>
  <name>myLogService</name>
  <stdout-config>myStdoutConfig</stdout-config>
  <logger-severity>Notice</logger-severity>
  <logger-severity-properties>
    <entry>
      <key>FileAdapter</key>
      <value>Debug</value>
    </entry>
    <entry>
      <key>CQLProcessor</key>
      <value>Debug</value>
    </entry>
  </logger-severity-properties>
</logging-service>
```

In the example, the `logging-service` element's unique identifier is `myLogService`.

F.25 message-filters

Use this element to configure one or more message filters for a [pubsub-bean](#) element.

F.25.1 Child Elements

The `message-filters` server configuration element contains one or more element child elements that each contain a `message-filter-name` and `message-filter-class` child element.

F.25.2 Attributes

The `message-filters` server configuration element has no attributes.

F.25.3 Example

The following example shows how to use the `message-filters` element in the Oracle CEP server configuration file:

```
<http-pubsub>
  <name>pubsub</name>
  <path>/pubsub</path>
  <pub-sub-bean>
    ...
    <message-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>
```

```

    </pub-sub-bean>
</http-pubsub>

```

F.26 name

Use this element to declare a unique identifier for an Oracle CEP server configuration element.

F.26.1 Child Elements

The name server configuration element has no child elements.

F.26.2 Attributes

The name server configuration element has no attributes.

F.26.3 Example

The following example shows how to use the name element in the Oracle CEP server configuration file:

```

<http-pubsub>
  <name>pubsub</name>
  <path>/pubsub</path>
  ...
</http-pubsub>

```

F.27 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.

F.27.1 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 name .
ssl-config-bean-name	string	The name of the SSL configuration object to use. If not null, then this client will create secure sockets using the specified SSL configuration. If not set, then no SSL will be supported.
provider-type	string	Specify which provider to use for the underlying socket implementation. For a list of the valid provider types, see "Network IO Providers" in the <i>Oracle Complex Event Processing Administrator's Guide</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.

F.27.2 Attributes

The `netio` server configuration element has no attributes.

F.27.3 Example

The following example shows how to use the `netio` element in the Oracle CEP server configuration file:

```
<netio>
  <name>myNetio</name>
  <port>12345</port>
</netio>
```

In the example, the `netio` element's unique identifier is `myNetio`.

F.28 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.

F.28.1 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
<code>name</code>	string	The name of this <code>netio</code> element. For more information, see name .
<code>ssl-config-bean-name</code>	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.
<code>provider-type</code>	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 Complex Event Processing Administrator's Guide</i> .

F.28.2 Attributes

The `netio-client` server configuration element has no attributes.

F.28.3 Example

The following example shows how to use the `netio-client` element in the Oracle CEP server configuration file:

```
<netio-client>
  <name>netiossl</name>
  <ssl-config-bean-name>sslConfig</ssl-config-bean-name>
  <provider-type>NIO</provider-type>
</netio-client>
```

In the example, the `netio-client` element's unique identifier is `netiossl`.

F.29 path

Use this element to configure the path for an [http-pubsub](#) element.

F.29.1 Child Elements

The `path` element has no child elements.

F.29.2 Attributes

The `path` element has no attributes.

F.29.3 Example

The following example shows how to use the `path` element in the Oracle CEP server configuration file:

```
<http-pubsub>
  <name>myPubsub</name>
  <path>/pubsub</path>
  <pub-sub-bean>
    <server-config>
      <supported-transport>
        <types>
          <element>long-polling</element>
        </types>
      </supported-transport>
      <publish-without-connect-allowed>
        true
      </publish-without-connect-allowed>
    </server-config>
    <channels>
      <element>
        <channel-pattern>/evsmonitor</channel-pattern>
      </element>
      <element>
        <channel-pattern>/evsalert</channel-pattern>
      </element>
      <element>
        <channel-pattern>/evsdomainchange</channel-pattern>
      </element>
    </channels>
  </pub-sub-bean>
</http-pubsub>
```

F.30 pubsub-bean

Use this element to configure a publish-subscribe bean for an `http-pubsub` element.

F.30.1 Child Elements

The `pubsub-bean` server configuration element supports the following child elements:

- `name`
- `server-config`
- `message-filters`
- `channels`
- `channel-constraints`
- `services`

F.30.2 Attributes

The `pubsub-bean` server configuration element has no attributes.

F.30.3 Example

The following example shows how to use the `pubsub-bean` element in the Oracle CEP server configuration file:

```
<http-pubsub>
  <name>myPubsub</name>
  <path>/pubsub</path>
  <pub-sub-bean>
    <server-config>
      <supported-transport>
        <types>
          <element>long-polling</element>
        </types>
      </supported-transport>
      <publish-without-connect-allowed>
        true
      </publish-without-connect-allowed>
    </server-config>
    <channels>
      <element>
        <channel-pattern>/evsmonitor</channel-pattern>
      </element>
      <element>
        <channel-pattern>/evsalert</channel-pattern>
      </element>
      <element>
        <channel-pattern>/evsdomainchange</channel-pattern>
      </element>
    </channels>
  </pub-sub-bean>
</http-pubsub>
```

F.31 rdbms-event-store-provider

Use this element to configure an event store provider that uses a relational database management system in the Oracle CEP server.

By default, Oracle CEP server uses a Berkeley database instance as the event store provider as [Section F.3, "bdb-config"](#) describes.

F.31.1 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
<code>name</code>	string	The name of this debug configuration. For more information, see name .
<code>init-timeout</code>	int	The maximum time (in milliseconds) that the Oracle CEP server will wait for this provider to initialize. Default: 10000 ms.

Table F–20 (Cont.) Child Elements of: rdbms-event-store-provider

XML Tag	Type	Description
data-source-name	string	The name of a data source element. For more information, see data-source .
user-policy-attributes	See Description	One or more entry child elements that each contain a <code>key</code> and <code>value</code> child element that you use to specify additional data source properties.

F.31.2 Attributes

The `rdbms-event-store-provider` server configuration element has no attributes.

F.31.3 Example

The following example shows how to use the `rdbms-event-store-provider` element in the Oracle CEP server configuration file:

```
<rdbms-event-store-provider>
  <name>test-rdbms-provider</name>
  <init-timeout>10000</init-timeout>
  <data-source-name>derby1</data-source-name>
  <user-policy-attributes>
    <entry>
      <key>key1</key>
      <value>value1</value>
    </entry>
    <key>key1</key>
    <value>value1</value>
  </user-policy-attributes>
</rdbms-event-store-provider>
```

In the example, the `rdbms-event-store-provider` element's unique identifier is `test-rdbms-provider`.

F.32 rmi

Use this element to configure an RMI service, which allows server-side objects to be exported to remote clients.

F.32.1 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 <code>rmi</code> element. For more information, see name .
heartbeat-period	int	The number of failed heartbeat attempts before triggering disconnect notifications to all registered listeners. Default-Value: 4.
http-service-name	string	The name of the HTTP service that this service should use to register remote objects. The service may be provided by a Jetty or Tomcat instance of the same name.

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.

F.32.2 Attributes

The `rmi` server configuration element has no attributes.

F.32.3 Example

The following example shows how to use the `rmi` element in the Oracle CEP server configuration file:

```
<rmi>
  <name>myRMI</name>
  <http-service-name>TestJetty</http-service-name>
</rmi>
```

In the example, the `rmi` element's unique identifier is `myRMI`.

F.33 scheduler

Use this element to configure `cql` scheduler options in the Oracle CEP server.

F.33.1 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 CEP Service Engine scheduling algorithm. This class determines in what order the Oracle CEP Service Engine scheduler executes Oracle CQL queries. Valid values: <ul style="list-style-type: none"> ▪ <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. ▪ <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. Default: <code>oracle.cep.execution.scheduler.RoundRobinScheduler</code>
runtime	long	Total number of seconds that the Oracle CEP Service Engine scheduler will run. Default: 1000000 ms.

Table F-22 (Cont.) Child Elements of: scheduler

XML Tag	Type	Description
time-slice	int	The frequency at which the Oracle CEP Service Engine scheduler executes Oracle CQL queries. Default: 1000 ms
schedule-on-new-thread	boolean	Whether or not the Oracle CEP Service Engine scheduler will use a separate thread. Options are: <ul style="list-style-type: none"> ▪ true: the scheduler runs in a separate thread. ▪ false: the scheduler runs in the same thread as the Oracle CEP Service Engine (Default).

F.33.2 Attributes

The `scheduler` server configuration element has no attributes.

F.33.3 Example

The following example shows how to use the `scheduler` element in the Oracle CEP server configuration file:

```
<cql>
  <name>myCQL</name>
  <scheduler>
    <class-name>oracle.cep.execution.scheduler.FIFOScheduler</class-name>
  </scheduler>
</cql>
```

F.34 server-config

Use this element to configure the server-specific properties of a [pubsub-bean](#) element.

F.34.1 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 name .

Table F–23 (Cont.) Child Elements of: server-config

XML Tag	Type	Description
supported-transport	See Description	<p>This element contains one or more <code>types</code> child elements, one for each supported transport. Each <code>types</code> child element contains an <code>element</code> child element with the transport name as a <code>string</code> value. Valid values:</p> <ul style="list-style-type: none"> <code>long-polling</code>: Using this transport, the client requests information from Oracle CEP server and if Oracle CEP server does not have information available, it does not reply until it has. When the Oracle CEP server replies, the client typically sends another request immediately. <code>callback-polling</code>: Use this transport for HTTP publish-subscribe applications using a cross domain configuration in which the browser downloads the page from one Web server (including the Javascript code) and connects to another server as an HTTP publish-subscribe client. This is required by the Bayeaux protocol. For more information on the Bayeaux protocol, see http://svn.xantus.org/shortbus/trunk/bayeux/bayeux.html. <p>For more information, see "How the HTTP Pub-Sub Server Works" in the <i>Oracle Complex Event Processing Administrator's Guide</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 <code>client-timeout-secs</code>. 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-milliseconds	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 work-manager.</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"> <code>true</code> <code>false</code>

F.34.2 Attributes

The `server-config` server configuration element has no attributes.

F.34.3 Example

The following example shows how to use the `server-config` element in the Oracle CEP server configuration file:

```
<http-pubsub>
  <name>pubsub</name>
  <path>/pubsub</path>
```

```

<pub-sub-bean>
  <server-config>
    <name>/pubsub</name>
    <supported-transport>
      <types>
        <element>long-polling</element>
      </types>
    </supported-transport>
    <publish-without-connect-allowed>true</publish-without-connect-allowed>
  </server-config>
<channels>
  <element>
    <channel-pattern>/evsmonitor</channel-pattern>
  </element>
  <element>
    <channel-pattern>/evsalert</channel-pattern>
  </element>
  <element>
    <channel-pattern>/evsdomainchange</channel-pattern>
  </element>
</channels>
</pub-sub-bean>
</http-pubsub>

```

F.35 services

Use this element to configure the service properties of a [pubsub-bean](#) element.

F.35.1 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> .

F.35.2 Attributes

The `services` server configuration element has no attributes.

F.35.3 Example

The following example shows how to use the `services` element in the Oracle CEP server configuration file:

```

<http-pubsub>
  <name>pubsub</name>
  <path>/pubsub</path>
  <pub-sub-bean>
    <server-config>
      <name>/pubsub</name>
      <supported-transport>

```

```

        <types>
          <element>long-polling</element>
        </types>
      </supported-transport>
      <publish-without-connect-allowed>true</publish-without-connect-allowed>
    </server-config>
  <channels>
    <element>
      <channel-pattern>/evsmonitor</channel-pattern>
    </element>
    <element>
      <channel-pattern>/evsalert</channel-pattern>
    </element>
    <element>
      <channel-pattern>/evsdomainchange</channel-pattern>
    </element>
  </channels>
  <services>
    <element>
      <service-channel>Foo</service-channel>
      <service-class>Foo</service-class>
      <service-method>Foo</service-method>
    </element>
  </services>
</pub-sub-bean>
</http-pubsub>

```

F.36 show-detail-error-message

Use this element to configure whether or not the Oracle CEP server uses secure connections.

F.36.1 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 name .
<code>value</code>	boolean	Whether or not to show detailed error messages. Valid values: <ul style="list-style-type: none"> ▪ <code>true</code>: the Oracle CEP server shows detailed error messages. ▪ <code>false</code>: the Oracle CEP server shows abbreviated error messages (default).

F.36.2 Attributes

The `show-detail-error-message` server configuration element has no attributes.

F.36.3 Example

The following example shows how to use the `show-detail-error-message` element in the Oracle CEP server configuration file:

```

<show-detail-error-message>
  <name>myShowDetail</name>
  <value>true</value>

```



```
</show-detail-error-message>
```

In the example, the `show-detail-error-message` element's unique identifier is `myShowDetail`.

F.37 ssl

Use this element to configure Secure Sockets Layer-specific properties on the Oracle CEP server.

F.37.1 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 name .
<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 <code>string</code> 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 <code>string</code> 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 CEP server uses a Federal Information Processing Standards (FIPS)-certified pseudo-random number generator. For more information, see "FIPS" in the <i>Oracle Complex Event Processing Administrator's Guide</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 <code>string</code> 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"> ▪ <code>FIPS186PRNG</code>
<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"> ▪ <code>JsafeJCE</code>

F.37.2 Attributes

The `ssl` server configuration element has no attributes.

F.37.3 Example

The following example shows how to use the `ssl` element in the Oracle CEP server configuration file:

```
<ssl>
  <name>sslConfig</name>
  <key-store>./ssl/evsidentity.jks</key-store>
  <key-store-pass>
    <password>{Salted-3DES}s4YUEvH4Wl2DAjb45iJnrw==</password>
  </key-store-pass>
  <key-store-alias>evsidentity</key-store-alias>
  <key-manager-algorithm>SunX509</key-manager-algorithm>
  <ssl-protocol>TLS</ssl-protocol>
  <enforce-fips>false</enforce-fips>
  <need-client-auth>false</need-client-auth>
</ssl>
```

In the example, the `ssl` element's unique identifier is `sslConfig`.

F.38 timeout-seconds

Use this element to configure `weblogic-jta-gateway` default transaction timeout in seconds in the Oracle CEP server.

Default: 60.

F.38.1 Child Elements

The `timeout-seconds` server configuration element has no attributes.

F.38.2 Attributes

The `timeout-seconds` server configuration element has no attributes.

F.38.3 Example

The following example shows how to use the `timeout-seconds` element in the Oracle CEP server configuration file:

```
<weblogic-jta-gateway>
  <name>myJTAGateway</name>
  <timeout-seconds>90</timeout-seconds>
  <weblogic-instances>
    <weblogic-instance>
      <domain-name>ocep_domain</domain-name>
      <server-name>fxserver</server-name>
      <protocol>t3</protocol>
      <host-address>ariel</host-address>
      <port>9002</port>
    </weblogic-instance>
  </weblogic-instances>
</weblogic-jta-gateway>
```

F.39 transaction-manager

Use this element to configure transaction manager properties in the Oracle CEP server.

F.39.1 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
<code>name</code>	string	The name of this <code>transaction-manager</code> element. For more information, see name .
<code>max-resource-requests-on-server</code>	int	Maximum number of concurrent requests to resources allowed for each server. Default: 50.
<code>max-resource-unavailable-millis</code>	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.
<code>security-interop-mode</code>	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"> ▪ default: The transaction coordinator makes calls using the kernel identity over an admin channel if it is enabled, and anonymous otherwise. Man-in-the-middle attacks are possible if the admin channel is not enabled. ▪ Performance: The transaction coordinator makes calls using anonymous at all times. This implies a security risk since a malicious third party could then try to affect the outcome of transactions using a man-in-the-middle attack. ▪ Compatibility: The transaction coordinator makes calls as the kernel identity over an insecure channel. This is a high security risk because a successful man-in-the-middle attack would allow the attacker to gain administrative control over both domains. This setting should only be used when strong network security is in place. Default: <code>default</code> .
<code>parallel-xa-enabled</code>	boolean	Execute XA calls in parallel if there are available threads. Default: <code>true</code> .
<code>tlog-location</code>	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.
<code>max-xa-call-millis</code>	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.
<code>timeout-seconds</code>	int	The default transaction timeout in seconds. Default: 30.
<code>checkpoint-interval-seconds</code>	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 <code>forget</code> 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: <code>true</code> .
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 <code>unregisterResource</code> 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 <code>unregisterResource</code> call returns immediately. At the end of the grace period, if there are still outstanding transactions associated with the resource, the <code>unregisterResource</code> call returns and a log message is written on the server on which the resource was previously registered. Default: 30.
rmi-service-name	string	The name of the RMI service that is used for distributed transaction coordination. For more information, see rmi .
max-unique-name-statistics	int	The maximum number of unique transaction names for which statistics will be maintained. Default: 1000.
purge-resource-from-checkpoint-interval-seconds	int	The interval that a particular resource must be accessed within for it to be included in the checkpoint record. Default: 86400.
max-transactions	int	The maximum number of simultaneous in-progress transactions allowed on this server. Default: 10000.
migration-checkpoint-interval-seconds	int	The interval that the checkpoint is done for the migrated transaction logs (TLOGs). Default: 60.

Table F-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.

F.39.2 Attributes

The `transaction-manager` server configuration element has no attributes.

F.39.3 Example

The following example shows how to use the `transaction-manager` element in the Oracle CEP server configuration file:

```
<transaction-manager>
  <name>My_tm</name>
  <timeout-seconds>30</timeout-seconds>
  <abandon-timeout-seconds>86400</abandon-timeout-seconds>
  <forget-heuristics>true</forget-heuristics>
  <before-completion-iteration-limit>12</before-completion-iteration-limit>
  <max-transactions>10100</max-transactions>
  <max-unique-name-statistics>500</max-unique-name-statistics>
  <max-resource-requests-on-server>50</max-resource-requests-on-server>
  <max-resource-unavailable-millis>1800000</max-resource-unavailable-millis>
  <recovery-threshold-millis>300000</recovery-threshold-millis>
  <max-transactions-health-interval-millis>
    60000
  </max-transactions-health-interval-millis>
  <purge-resource-from-checkpoint-interval-seconds>
    86400
  </purge-resource-from-checkpoint-interval-seconds>
  <checkpoint-interval-seconds>300</checkpoint-interval-seconds>
  <parallel-xa-enabled>true</parallel-xa-enabled>
  <unregister-resource-grace-period>30</unregister-resource-grace-period>
  <security-interop-mode>default</security-interop-mode>
  <rmi-service-name>RMI_cel</rmi-service-name>
</transaction-manager>
```

In the example, the `transaction-manager` element's unique identifier is `My_tm`.

F.40 use-secure-connections

Use this element to configure whether or not the Oracle CEP server uses secure connections.

For more information, see "How to Configure SSL in a Multi-Server Domain for Oracle CEP Visualizer" in the *Oracle Complex Event Processing Administrator's Guide*.

F.40.1 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 name .
<code>value</code>	boolean	Whether or not to use secure connections. Valid values: <ul style="list-style-type: none"> ▪ <code>true</code>: the Oracle CEP server uses only secure connections. ▪ <code>false</code>: the Oracle CEP server accepts connections that are not secure.

F.40.2 Attributes

The `use-secure-connections` server configuration element has no attributes.

F.40.3 Example

The following example shows how to use the `use-secure-connections` element in the Oracle CEP server configuration file:

```
<use-secure-connections>
  <name>myUseSecConn</name>
  <value>true</value>
</use-secure-connections>
```

In the example, the `use-secure-connections` element's unique identifier is `myUseSecConn`.

F.41 weblogic-instances

Use this element to configure Oracle CEP server instances for a [weblogic-jta-gateway](#) element.

F.41.1 Child Elements

The `weblogic-instances` server configuration element supports zero or more `weblogic-instance` child elements that each contain the child elements that [Table F-29](#) lists.

Table F-29 Child Elements of: `weblogic-instances`

XML Tag	Type	Description
<code>domain-name</code>	string	Specifies the name of the domain of the Oracle CEP server.
<code>server-name</code>	string	Specifies the name of the Oracle CEP 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 CEP server.
<code>port</code>	int	The netio port for the Oracle CEP server.

F.41.2 Attributes

The `weblogic-instances` server configuration element has no attributes.

F.41.3 Example

The following example shows how to use the `weblogic-instances` element in the Oracle CEP server configuration file:

```
<weblogic-jta-gateway>
  <name>myJTAGateway</name>
  <timeout-seconds>90</timeout-seconds>
  <weblogic-instances>
    <weblogic-instance>
      <domain-name>ocep_domain</domain-name>
      <server-name>fxserver</server-name>
      <protocol>t3</protocol>
      <host-address>ariel</host-address>
      <port>9002</port>
    </weblogic-instance>
  </weblogic-instances>
</weblogic-jta-gateway>
```

F.42 weblogic-jta-gateway

Use this element to configure the attributes for the singleton Oracle CEP server client JTA gateway service.

F.42.1 Child Elements

The `weblogic-jta-gateway` server configuration element supports the following child elements:

- `name`
- `timeout-seconds`
- `weblogic-instances`

F.42.2 Attributes

The `weblogic-jta-gateway` server configuration element has no attributes.

F.42.3 Example

The following example shows how to use the `weblogic-jta-gateway` element in the Oracle CEP server configuration file:

```
<weblogic-jta-gateway>
  <name>myJTAGateway</name>
  <timeout-seconds>90</timeout-seconds>
  <weblogic-instances>
    <weblogic-instance>
      <domain-name>ocep_domain</domain-name>
      <server-name>fxserver</server-name>
      <protocol>t3</protocol>
      <host-address>ariel</host-address>
      <port>9002</port>
    </weblogic-instance>
  </weblogic-instances>
</weblogic-jta-gateway>
```

In the example, the `weblogic-jta-gateway` element's unique identifier is `myJTAGateway`.

F.43 weblogic-rmi-client

Use this element to configure the attributes for the singleton Oracle CEP server RMI client.

F.43.1 Child Elements

The `weblogic-rmi-client` server configuration element supports the child elements that [Table F-30](#) lists.

Table F-30 Child Elements of: *weblogic-rmi-client*

XML Tag	Type	Description
<code>name</code>	string	The name of this <code>weblogic-rmi-client</code> element. For more information, see name .
<code>netio-name</code>	string	Specifies the name of the <code>netio-client</code> element to use. For more information, see netio-client .
<code>secure-netio-name</code>	string	Specifies the name of the <code>netio-client</code> element configured for SSL. For more information, see netio-client .

F.43.2 Attributes

The `weblogic-rmi-client` server configuration element has no attributes.

F.43.3 Example

The following example shows how to use the `weblogic-rmi-client` element in the Oracle CEP server configuration file:

```
<netio-client>
  <name>netio</name>
  <provider-type>NIO</provider-type>
</netio-client>

<netio-client>
  <name>netiossl</name>
  <provider-type>NIO</provider-type>
  <ssl-config-bean-name>sslConfig</ssl-config-bean-name>
</netio-client>

<weblogic-rmi-client>
  <name>wlclient</name>
  <netio-name>netio</netio-name>
  <secure-netio-name>netiossl</secure-netio-name>
</weblogic-rmi-client>
```

In the example, the `weblogic-rmi-client` element's unique identifier is `wlclient`.

F.44 work-manager

Use this element to configure a work manager on the Oracle CEP server.

F.44.1 Child Elements

The `work-manager` server configuration element supports the child elements that [Table F-31](#) lists.

Table F-31 Child Elements of: work-manager

XML Tag	Type	Description
name	string	The name of this <code>work-manager</code> element. For more information, see name .
min-threads-constraint	int	The minimum threads constraint this work manager should use. Default: -1.
fairshare	int	The fairshare value this work manager should use. Default: -1.
max-threads-constraint	int	The maximum threads constraint this work manager should use. Default: -1.

F.44.2 Attributes

The `work-manager` server configuration element has no attributes.

F.44.3 Example

The following example shows how to use the `work-manager` element in the Oracle CEP server configuration file:

```
<work-manager>
  <name>WM</name>
  <fairshare>5</fairshare>
  <min-threads-constraint>1</min-threads-constraint>
  <max-threads-constraint>4</max-threads-constraint>
</work-manager>
```

In the example, the `work-manager` element's unique identifier is `WM`.

F.45 xa-params

Use this element to specify distributed transaction-related [data-source](#) parameters.

F.45.1 Child Elements

The `xa-params` server configuration element supports the child elements that [Table F-32](#) lists.

Table F-32 Child Elements of: xa-params

XML Tag	Type	Description
keep-xa-conn-till-tx-complete	boolean	Enables the server to associate the same XA database connection from the connection pool with a global transaction until the transaction completes. Only applies to connection pools that use an XA driver. Use this setting to work around specific problems with JDBC XA drivers. Default: <code>true</code> .

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> .

Table F–32 (Cont.) Child Elements of: xa-params

XML Tag	Type	Description
xa-retry-interval-seconds	int	The number of seconds between XA retry operations if XARetryDurationSeconds is set to a positive value. Default: 60.
recover-only-once	boolean	Specifies that the transaction manager calls recover on the resource only once. Only applies to data sources that use an XA driver. Use this setting to work around specific problems with JDBC XA drivers. Default: false.
need-tx-ctx-on-close	boolean	Specifies whether the XA driver requires a distributed transaction context when closing various JDBC objects (result sets, statements, connections, and so forth). Only applies to connection pools that use an XA driver. When enabled, SQL exceptions that are thrown while closing the JDBC objects without a transaction context will be suppressed. Use this setting to work around specific problems with JDBC XA drivers. Default: false.

F.45.2 Attributes

The xa-params server configuration element has no attributes.

F.45.3 Example

The following example shows how to use the xa-params element in the Oracle CEP server configuration file:

```
<data-source>
  <name>orads</name>
  <xa-params>
    <keep-xa-conn-till-tx-complete>true</keep-xa-conn-till-tx-complete>
  </xa-params>
  <driver-params>
    <url>jdbc:oracle:thin:@localhost:1521:ce102</url>
    <driver-name>oracle.jdbc.OracleDriver</driver-name>
    <properties>
      <element>
        <name>user</name>
        <value>wlevs</value>
      </element>
      <element>
        <name>password</name>
        <value>wlevs</value>
      </element>
    </properties>
  </driver-params>
  <connection-pool-params>
    <initial-capacity>5</initial-capacity>
    <max-capacity>10</max-capacity>
    <test-table-name>SQL SELECT 1 FROM DUAL</test-table-name>
    <test-frequency-seconds>5</test-frequency-seconds>
  </connection-pool-params>
  <data-source-params>
    <jndi-names>
      <element>orads</element>
    </jndi-names>
    <global-transactions-protocol>None</global-transactions-protocol>
  </data-source-params>
</data-source>
```

Schema Reference: Message Catalog msgcat.dtd

This appendix describes the elements of the `msgcat.dtd` schema.

For more information, see:

- [Appendix H, "Schema Reference: Locale Message Catalog l10n_msgcat.dtd"](#)
- [Section 24.4, "Managing Log Message Catalogs"](#)

G.1 Overview of the Message Catalog Elements

Oracle CEP provides a number of message catalog elements that you use to define log messages that you can localize.

G.1.1 Element Hierarchy

The top-level Oracle CEP 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
```

G.1.2 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>
</message_catalog>

```

G.2 message_catalog

This element represents the log message catalog.

G.2.1 Child Elements

The message_catalog element supports the following child elements:

- [message](#)
- [logmessage](#)

G.2.2 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"> ▪ <code>l10n_package.mycatLogLocalizer.properties</code> ▪ <code>l10n_packagemycatLogLocalizerDetail.properties</code> 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"> ▪ 000000 for Oracle CEP server catalogs ▪ 500000 for user-defined catalogs 	String	No.
endid	Specifies the highest message ID used in this catalog. Syntax: one to six decimal digits. Example: <code>endid="600100"</code> Valid values are: <ul style="list-style-type: none"> ▪ 499999 for Oracle CEP server catalogs ▪ 999999 for user-defined catalogs 	String	No.
loggables	Indicates whether to generate additional methods that return loggable objects. Example: <code>loggable="true"</code> Valid values are: <ul style="list-style-type: none"> ▪ true ▪ false Default: <code>false</code>	String	

Table G-1 (Cont.) Attributes of the message_catalog Element

Attribute	Description	Data Type	Required?
prefix	<p>Specifies a String to be prepended to message IDs when logged. Oracle CEP 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:</p> <pre><[prefix-]id></pre> <p>Where <code>prefix</code> is this attribute and <code>id</code> is the six-digit message ID associated with a specific message.</p> <p>For example, if <code>prefix</code> is "XYZ", then message 987654 would be shown in a log entry as <code><XYZ-987654></code>. If <code>prefix</code> is not defined, then the log entry would be <code><987654></code>.</p> <p>Syntax: any String (should be limited to five characters).</p> <p>Example: <code>prefix="CEP"</code></p> <p>Valid values are:</p> <ul style="list-style-type: none"> ■ "CEP" for Oracle CEP server catalogs. ■ null for user-defined catalogs 	String	No.
description	<p>An optional attribute that serves to document the catalog content.</p> <p>Example: <code>description="Contains messages logged by the foobar application"</code></p>	String	No.

G.2.3 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>
```



```

    </action>
  </logmessage>
</message_catalog>

```

G.3 logmessage

Use this element to define a formal log message.

G.3.1 Child Elements

The logmessage component configuration element supports the following child elements:

- [messagebody](#)
- [messagedetail](#)
- [cause](#)
- [action](#)

G.3.2 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 <code>baseid</code> and <code>endid</code> attributes.</p> <p>Use: Value must be in the range defined by the <code>baseid</code> and <code>endid</code> attributes defined in the <code>message_catalog</code> element.</p> <p>Syntax: one to six decimal digits.</p> <p>Example: <code>messageid="600001"</code></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: <code>Long.toString(new Date().getTime());</code></p>	String	No.
severity	<p>Indicates the severity of the log message. User-defined catalogs may only use <code>debug</code>, <code>info</code>, <code>warning</code>, and <code>error</code>.</p> <p>Valid values:</p> <ul style="list-style-type: none"> ▪ <code>error</code> ▪ <code>warning</code> ▪ <code>info</code> ▪ <code>debug</code> <p>Example: <code>severity="warning"</code></p>	String	Yes.

Table G-2 (Cont.) Attributes of the logmessage Element

Attribute	Description	Data Type	Required?
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 Section G.3.1, "Child Elements" 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"> ▪ Logger methods format the message body into the default locale and log the results. ▪ Getter methods return the message body prefixed by the <code>subsystem</code> and <code>messageid</code>, as follows: <code>[subsystem:msgid]text</code>. <p>Valid values are:</p> <ul style="list-style-type: none"> ▪ <code>logger</code> ▪ <code>getter</code> <p>Default: <code>logger</code>.</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"> ▪ <code>true</code> ▪ <code>false</code> <p>Default: <code>true</code>.</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"> ▪ <code>true</code> ▪ <code>false</code> <p>Default: <code>false</code>.</p>	String	No.

G.3.3 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>

```

G.4 message

Use this element to define an informal log message.

G.4.1 Child Elements

The message element supports the following child elements:

- [messagebody](#)

G.4.2 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.
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.

Table G-3 (Cont.) Attributes of the message Element

Attribute	Description	Data Type	Required?
method	<p>Method signature for formatting this message.</p> <p>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.</p> <p>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>.</p> <p>Example: <code>method="getNoAuthorization(String filename, java.util.Date creDate)"</code></p> <p>This example would result in a method in the <code>TextFormatter</code> class as follows:</p> <pre>public String getNoAuthorization(String filename, java.util.Date creDate)</pre>	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"> ▪ true ▪ false <p>Default: false.</p>	String	No.

G.4.3 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>
```

G.5 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 G.5.3, "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`

G.5.1 Child Elements

The `messagebody` element has no child elements.

G.5.2 Attributes

The `messagebody` element has no attributes.

G.5.3 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>
```

G.6 messagedetail

Use this element to define a detailed description of the event. This element may contain any argument place holders.

Syntax: a String

G.6.1 Child Elements

The `messagedetail` element supports has no child elements.

G.6.2 Attributes

The `messagedetail` element has no attributes.

G.6.3 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>
```

G.7 cause

Use this element to define the root cause of the problem. This element can contain any argument place holders.

Syntax: a String.

G.7.1 Child Elements

The `cause` element supports has no child elements.

G.7.2 Attributes

The `cause` element has no attributes.

G.7.3 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>
```

G.8 action

Use this element to define the recommended resolution. This element can contain any argument place holders.

Syntax: a `String`.

G.8.1 Child Elements

The `action` element supports has no child elements.

G.8.2 Attributes

The `action` element has no attributes.

G.8.3 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 describes the elements of the `l10n_msgcat.dtd` schemas.

For more information, see:

- [Appendix G, "Schema Reference: Message Catalog msgcat.dtd"](#)
- [Section 24.4, "Managing Log Message Catalogs"](#)

H.1 Overview of the Locale Message Catalog Elements

Oracle CEP provides a number of locale message catalog elements that you use to define log messages localized for a given Java locale.

H.1.1 Element Hierarchy

The top-level Oracle CEP 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
```

H.1.2 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>
```

H.2 locale_message_catalog

Use this element to define values for a parameterized Oracle CQL or EPL rule in an EPL processor component.

H.2.1 Child Elements

The locale_message_catalog element supports the following child elements:

- [message](#)
- [logmessage](#)

H.2.2 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?
<code>l10n_package</code>	Java package containing generated <code>LogLocalizer</code> or <code>TextLocalizer</code> properties for this catalog. Properties file are named after the catalog file name. For example, for a French log message catalog called <code>mycat.xml</code> , a properties file called <code><l10n_package>.mycatLogLocalizer_fr_FR.properties</code> is generated.	String	No.
<code>version</code>	Specifies the version of the <code>l10n_msgcat.dtd</code> 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.

H.2.3 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/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>
```

H.3 logmessage

Use this element to define a formal log message.

H.3.1 Child Elements

The `logmessage` component configuration element supports the following child elements:

- [messagebody](#)

- [messagedetail](#)
- [cause](#)
- [action](#)

H.3.2 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.

H.3.3 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/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>
```

H.4 message

Use this element to define an informal log message.

H.4.1 Child Elements

The message element supports the following child elements:

- [messagebody](#)

H.4.2 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.

H.4.3 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>
```

H.5 messagebody

Use this element to define a concise, one-line log message body.

H.5.1 Child Elements

The messagebody element has no child elements.

H.5.2 Attributes

The messagebody element has no attributes.

H.5.3 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/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>
```

H.6 messagedetail

Use this element to define a more detailed explanation of the issue being logged.

H.6.1 Child Elements

The `messagedetail` element supports has no child elements.

H.6.2 Attributes

The `messagedetail` element has no attributes.

H.6.3 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/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>

```

H.7 cause

Use this element to define the root cause of the issue being logged.

H.7.1 Child Elements

The cause element supports has no child elements.

H.7.2 Attributes

The cause element has no attributes.

H.7.3 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>

```

```
</logmessage>
</message_catalog>
```

H.8 action

Use this element to define how to fix the issue being logged, if possible.

H.8.1 Child Elements

The `action` element supports has no child elements.

H.8.2 Attributes

The `action` element has no attributes.

H.8.3 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/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>
```

Oracle CEP Metadata Annotation Reference

This section contains information on the following subjects:

- [Section I.1, "Overview of Oracle CEP Metadata Annotations"](#)
- [Section I.2, "com.bea.wlevs.configuration.Activate"](#)
- [Section I.3, "com.bea.wlevs.configuration.Prepare"](#)
- [Section I.4, "com.bea.wlevs.configuration.Rollback"](#)
- [Section I.5, "com.bea.wlevs.util.Service"](#)

I.1 Overview of Oracle CEP Metadata Annotations

Oracle CEP metadata annotations are used to access the configuration of an Oracle CEP component. Oracle CEP offers the following annotations:

- [Section I.1.1, "Adapter Lifecycle Annotations"](#)
- [Section I.1.2, "OSGi Service Reference Annotations"](#)
- [Section I.1.3, "Resource Access Annotations"](#)

For more information, see:

- [Section 1.1.9, "Oracle CEP Application Lifecycle"](#)
- [Section 1.4, "Configuring Oracle CEP Resource Access"](#)

I.1.1 Adapter Lifecycle Annotations

You use the following annotations to specify the methods of an adapter Java implementation that handle various stages of the adapter's 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](#)

I.1.2 OSGi Service Reference Annotations

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

I.1.3 Resource Access Annotations

Use the following annotations to configure resource access at design time and the corresponding deployment XML to override this configuration at deploy time:

- `javax.annotation.Resource`

For more information, see [Section 1.4, "Configuring Oracle CEP Resource Access"](#).

I.2 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 CEP uses to send configuration information to the adapter.

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

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

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

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

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

This metadata annotation does not have any attributes.

I.2.1 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 CEP automatically creates an instance of this class when the application is deployed. For example, the adapter section of the `helloworldAdapter` configuration file is as follows:

```

<?xml version="1.0" encoding="UTF-8"?>
  <helloworld:config
    ...
    <adapter>
      <name>helloworldAdapter</name>
      <message>HelloWorld - the current time is:</message>
    </adapter>
  </helloworld:config>

```

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

I.3 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 CEP uses to send configuration information to the adapter.

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

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

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

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

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

This metadata annotation does not have any attributes.

I.3.1 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 CEP automatically creates an instance of this class when the application is deployed. In the HelloWorld example, the adapter configuration has been extended. See the example in [Appendix I.2, "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.

I.4 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 CEP uses to send configuration information to the adapter.

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

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

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

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

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

This metadata annotation does not have any attributes.

I.4.1 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 CEP automatically creates an instance of this class when the application is deployed. In the HelloWorld example, the adapter configuration has been extended. See the example in [Appendix I.2, "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.

I.5 com.bea.wlevs.util.Service

Target: Method

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

I.5.1 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?
serviceName	The name of the bean that backs the injected service. May be null.	String	No.
cardinality	Valid values for this attribute are: <ul style="list-style-type: none"> ▪ ServiceCardinality.C0__1 ▪ ServiceCardinality.C0__N ▪ ServiceCardinality.C1__1 ▪ ServiceCardinality.C1__N Default value is ServiceCardinality.C1__1.	enum	No.
contextClassLoader	Valid values for this attribute are: <ul style="list-style-type: none"> ▪ ServiceClassLoader.CLIENT ▪ ServiceClassLoader.SERVICE_PROVIDER ▪ ServiceClassLoader.UNMANAGED Default value is ServiceClassLoader.CLIENT.	enum	No.
timeout	Timeout for service resolution in milliseconds. Default value is 30000.	int	No.

Table I-1 (Cont.) Attributes of the com.bea.wlevs.util.Service JWS Annotation Tag

Name	Description	Data Type	Required?
serviceType	Interface (or class) of the service to be injected Default value is <code>Service.class</code> .	Class	No.
filter	Specifies the filter used to narrow service matches. Value may be <code>null</code> .	String	No.

I.5.2 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 2.7, "Accessing the Event Type Repository"](#).

Oracle CEP IDE for Eclipse Tutorial

This appendix provides a simple tutorial that illustrates the basics of creating, deploying, and debugging a simple Oracle CEP application.

This tutorial includes the following steps:

- [Appendix J.1, "Before You Begin"](#)
- [Appendix J.2, "Step 1: Create an Oracle CEP Definition"](#)
- [Appendix J.3, "Step 2: Create an Oracle CEP Application"](#)
- [Appendix J.4, "Step 3: Start the Oracle CEP Server and Deploy the Project"](#)
- [Appendix J.5, "Step 4: Change Code and Redeploy"](#)
- [Appendix J.6, "Step 5: Debug the Deployed Application"](#)
- [Appendix J.7, "Next Steps"](#)

J.1 Before You Begin

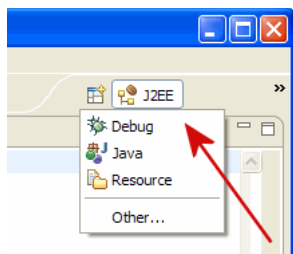
This tutorial assumes that you have installed an Oracle CEP server and Oracle CEP IDE for Eclipse on the local computer. For more information, see:

<http://www.oracle.com/technologies/soa/complex-event-processing.html>.

Before beginning this tutorial it is a good idea to switch to the Java or the Java EE perspective: both of which are good starting points for using Oracle CEP IDE for Eclipse to develop Oracle CEP applications. Although Oracle CEP applications are not Java EE applications, many of the same views are useful between the two.

To change the perspective, select **Window > Open Perspective** menu, or use the perspective shortcut menu as shown in [Figure J-1](#).

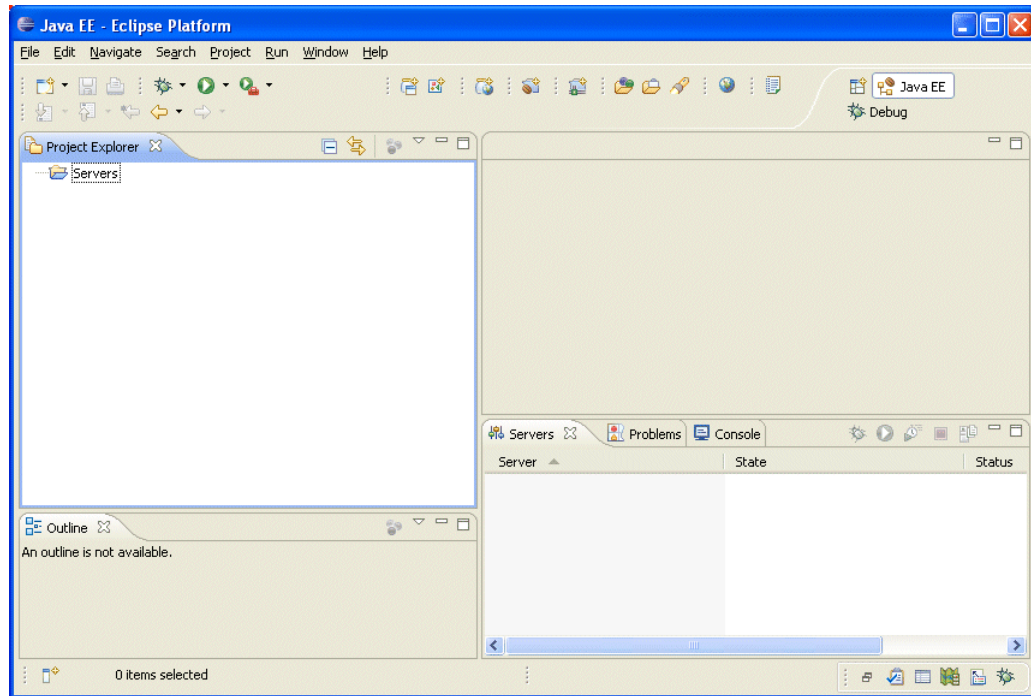
Figure J-1 Perspective Shortcut Menu



If you are in the Java perspective, you will want to open the Servers view by selecting **Window > Show View > Servers**.

Your Oracle CEP IDE for Eclipse should look like [Figure J-2](#).

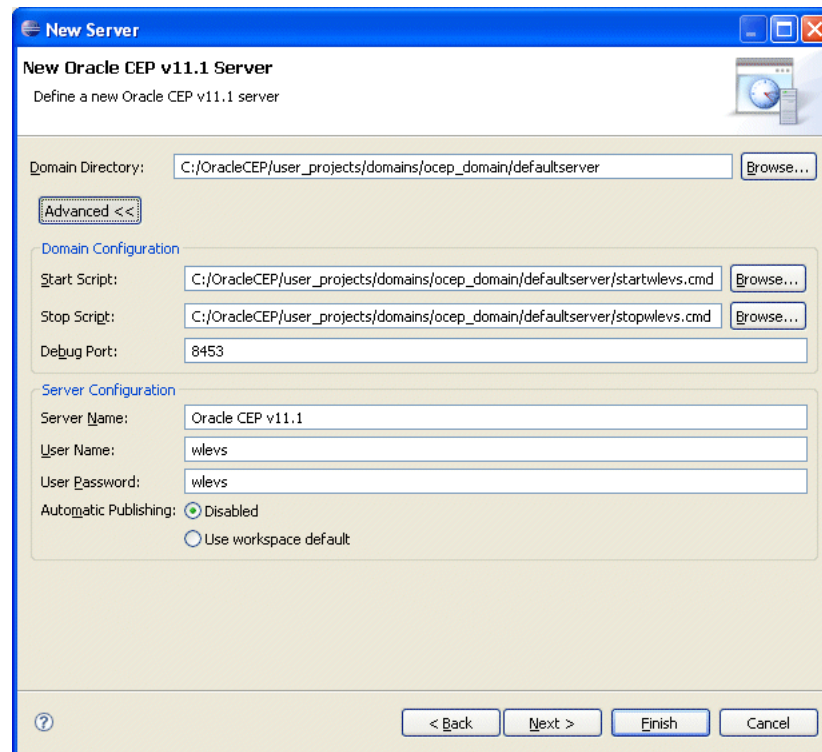
Figure J-2 Oracle CEP IDE for Eclipse



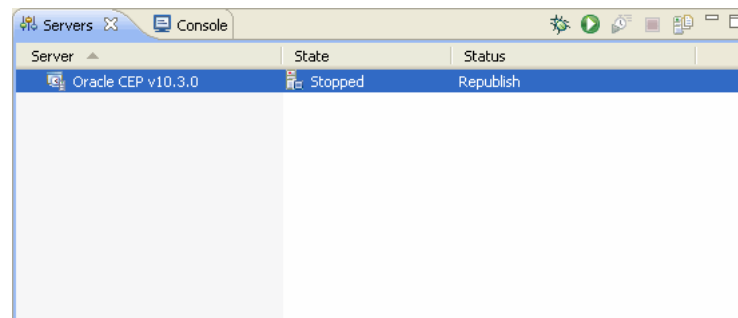
J.2 Step 1: Create an Oracle CEP Definition

The first step is to tell the Oracle CEP IDE for Eclipse where your Oracle CEP server is installed on the local machine. To do this, follow the steps in the [Section 5.2.1, "How to Create a Local Oracle CEP Server and Server Runtime"](#).

For this tutorial, enable automatic publishing on the server by selecting Use Workspace Default for the Automatic Publishing options on the second wizard page as [Figure J-3](#) shows.

Figure J-3 Configuring the Oracle CEP Server for Automatic Publishing

When finished, your Servers view should look as [Figure J-4](#) shows.

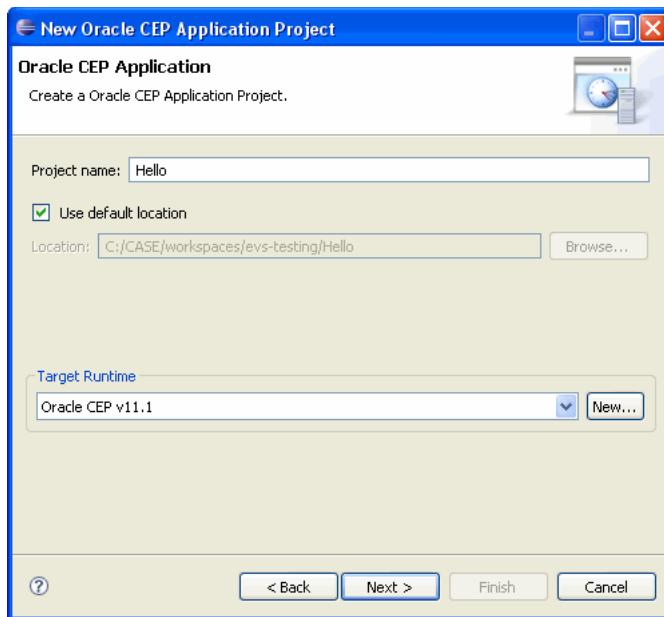
Figure J-4 Oracle CEP IDE for Eclipse Server View

J.3 Step 2: Create an Oracle CEP Application

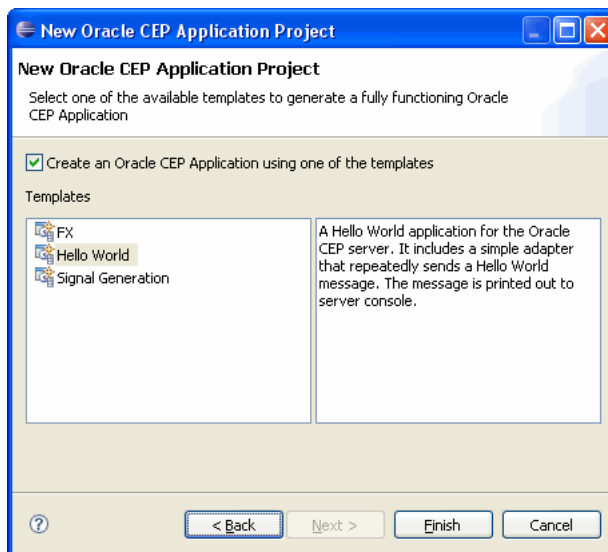
Next, create an Oracle CEP application. The basic the steps are described in [Section 5.2.1, "How to Create a Local Oracle CEP Server and Server Runtime"](#).

In this tutorial, select the **File > New > Other**, expand the **Oracle CEP** option and select **Oracle CEP Application Project**.

In the New Oracle CEP Application Project dialog, enter a project name of `Hello` and select the `Oracle CEP v10.3.0` the Target Runtime (created in [Appendix J.2, "Step 1: Create an Oracle CEP Definition"](#)) as [Figure J.5](#) shows.

Figure J-5 New Oracle CEP Application Project Dialog

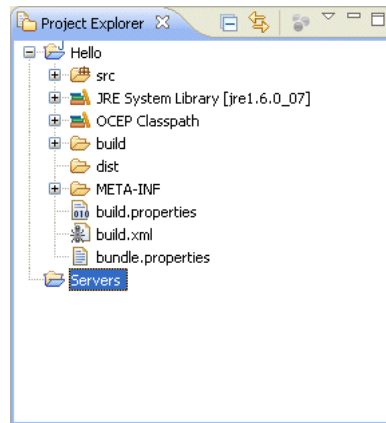
Click **Next**. Leave the Bundle Properties at their defaults and click **Next** again. The Templates dialog appears as [Figure J-6](#) shows.

Figure J-6 Templates Dialog

This page allows you to select what initial application files should be placed in the project. Check **Create an Oracle CEP Application using one of these templates** and select the **Hello World** template.

Click **Finish**.

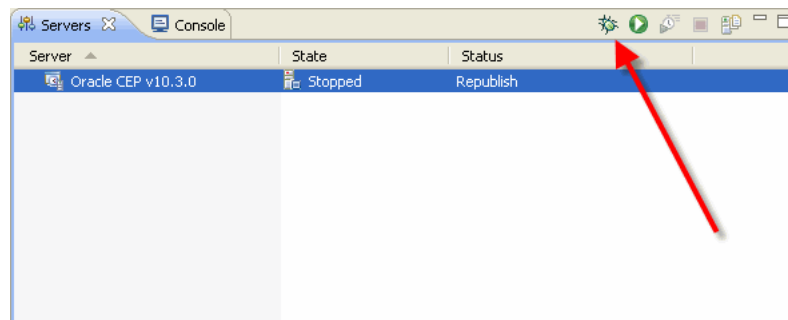
The Project Explorer should contain the files shown in [Figure J-7](#).

Figure J-7 Project Explorer

J.4 Step 3: Start the Oracle CEP Server and Deploy the Project

Now that the project is created and includes example content, it's time to start the server and deploy the project.

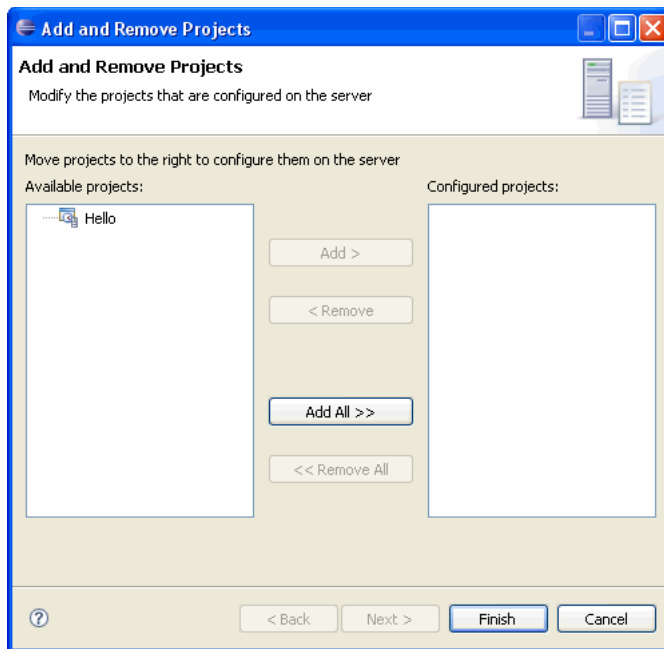
To start the server, click the **Debug** button on the Servers view as shown in [Figure J-8](#). The debug icon looks like a bug on the view toolbar.

Figure J-8 Starting the Oracle CEP Server

In the Console view, some Oracle CEP server log messages will scroll by, followed by the server indicating that it has started, such as:

```
<Jan 22, 2009 4:59:41 PM EST> <Notice> <Server> <BEA-2046000> <Server STARTED>
```

Next, deploy the project to the server. In the Servers view, right-click the server and select **Add/Remove Projects**. The Add and Remove Projects dialog appears as shown in [Figure J-9](#).

Figure J-9 Add and Remove Projects Dialog

Select the **Hello** project from the Available projects list on the left and click **Add** to move the project to the Configured projects list on the right.

Click **Finish**. The Oracle CEP IDE for Eclipse deploys the project to the server.

In the Console view, the Oracle CEP server indicates that the application is deployed, such as (lines are broken for readability):

```
<Jan 23, 2009 9:24:13 AM EST> <Notice> <Spring> <BEA-2047000>
  <The application context for "com.bea.wlevs.monitor" was deployed successfully>
<Jan 23, 2009 9:24:17 AM EST> <Notice> <Spring> <BEA-2047000>
  <The application context for "com.bea.wlevs.dataservices" was deployed successfully>
<Jan 23, 2009 9:26:06 AM EST> <Notice> <Server> <BEA-2045000>
  <The application bundle "Hello" was deployed successfully to
    file:/C:/OracleCEP/user_projects/domains/ocep_
domain/defaultserver/applications/Hello/Hello.jar
    with version 1232720766696>
<Jan 23, 2009 9:26:06 AM EST> <Notice> <Server> <BEA-2045000>
  <The application bundle "Hello" was deployed successfully to
    file [C:\OracleCEP\user_projects\domains\ocep_
domain\defaultserver\applications\Hello\Hello.jar]
    with version 1232720766696>
<Jan 23, 2009 9:26:22 AM EST> <Notice> <Spring> <BEA-2047000>
  <The application context for "Hello" was deployed successfully>
```

Because the application is now deployed and running, you should see messages from your application being logged to the Console view

```
Message: HelloWorld - the current time is:9:26:22 AM
Message: HelloWorld - the current time is:9:26:23 AM
Message: HelloWorld - the current time is:9:26:23 AM
...
```

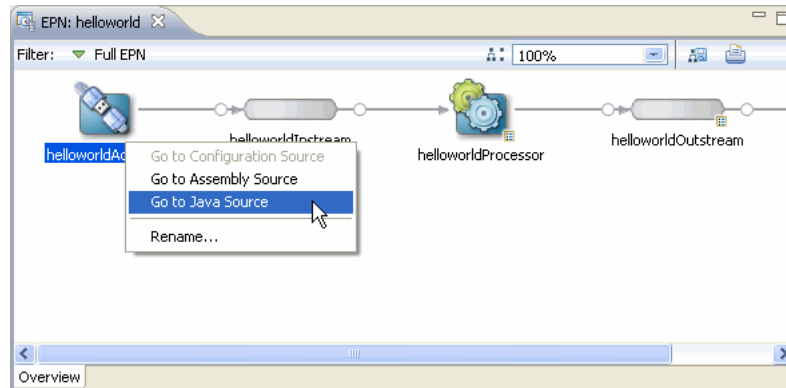
J.5 Step 4: Change Code and Redeploy

Next, we will change some code and redeploy the project to demonstrate the iterative development cycle.

Open the EPN (see [Section 6.1, "Opening the EPN Editor"](#)).

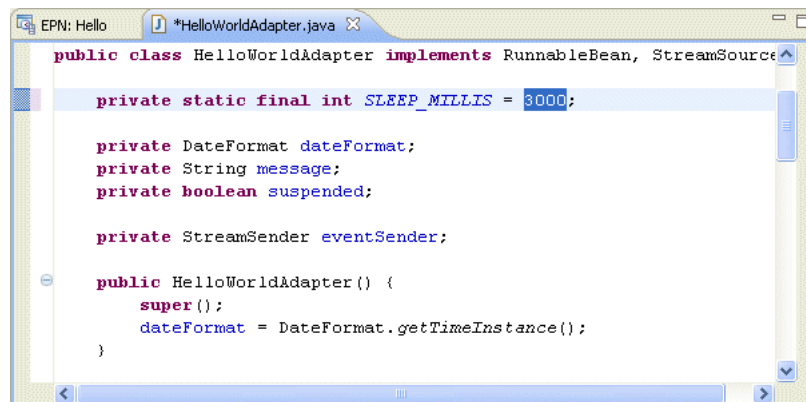
In the EPN Editor, right click on `helloworldAdapter` and select **Go to Java Source** as shown in [Figure J-10](#).

Figure J-10 *Editing Java Source for an Adapter*



The Java editor opens as [Figure J-11](#) shows.

Figure J-11 *Java Editor for helloworldAdapter*



Using the Java editor, change the value of the `SLEEP_MILLIS` field to 3000 so that events only fire every 3 seconds. Select **File > Save**.

Because the default setting is to automatically redeploy projects, no further action is required. Oracle CEP IDE for Eclipse will redeploy the project after a few seconds and the rate of HelloWorld messages in the Console view will slow down to every 3 seconds as shown below (lines are broken for readability).

```

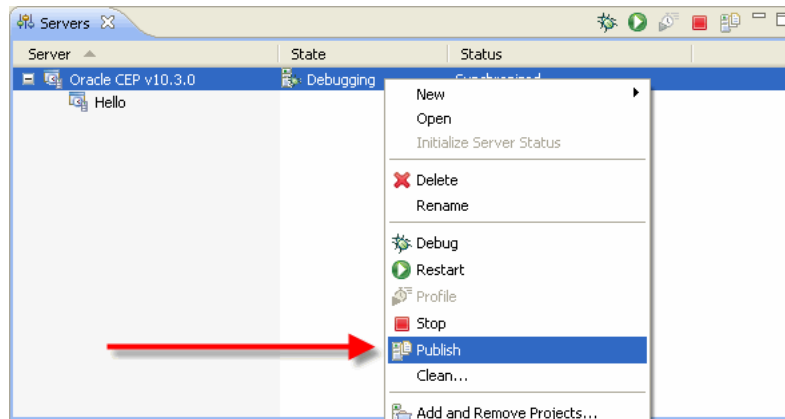
Message: HelloWorld - the current time is:9:37:41 AM
Message: HelloWorld - the current time is:9:37:41 AM
Message: HelloWorld - the current time is:9:37:41 AM
<Jan 23, 2009 9:37:42 AM EST> <Notice> <Spring> <BEA-2047001>
  <The application context "Hello" was undeployed successfully>
<Jan 23, 2009 9:37:42 AM EST> <Notice> <Server> <BEA-2045000>
  <The application bundle "Hello" was deployed successfully to
  file:/C:/OracleCEP/user_projects/domains/ocep_
  domain/defaultserver/applications/Hello/Hello.jar
  with version 1232721461898>
<Jan 23, 2009 9:37:43 AM EST> <Notice> <Spring> <BEA-2047000>
  <The application context for "Hello" was deployed successfully>
Message: HelloWorld - the current time is:9:37:43 AM

```

Message: HelloWorld - the current time is:9:37:46 AM

If automatic publishing is disabled, to republish the project, right-click on the server in the Servers view and select **Publish** as [Figure J-12](#) shows.

Figure J-12 Manually Publishing a Project



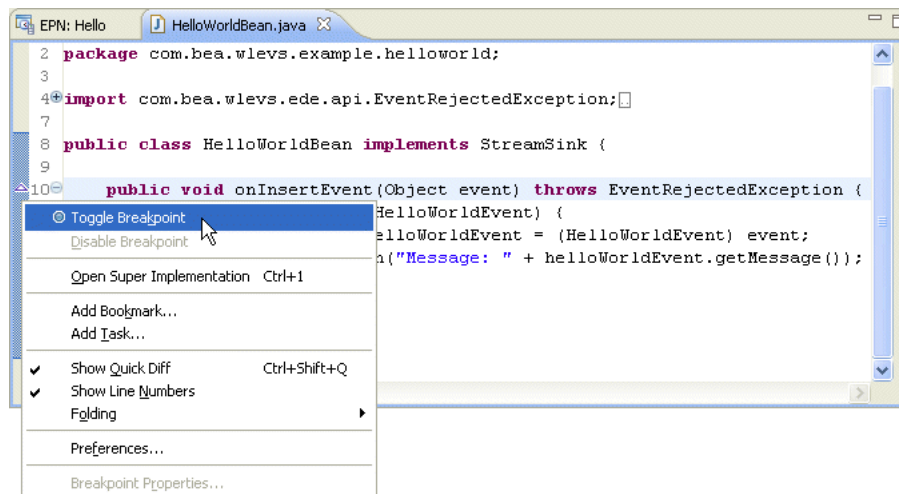
J.6 Step 5: Debug the Deployed Application

Next, we will debug some code now that the application is up and running.

Open the EPN (see [Section 6.1](#), "Opening the EPN Editor").

In the EPN Editor, right click on **HelloWorldBean** and select **Go to Java Source** as shown in [Figure J-13](#).

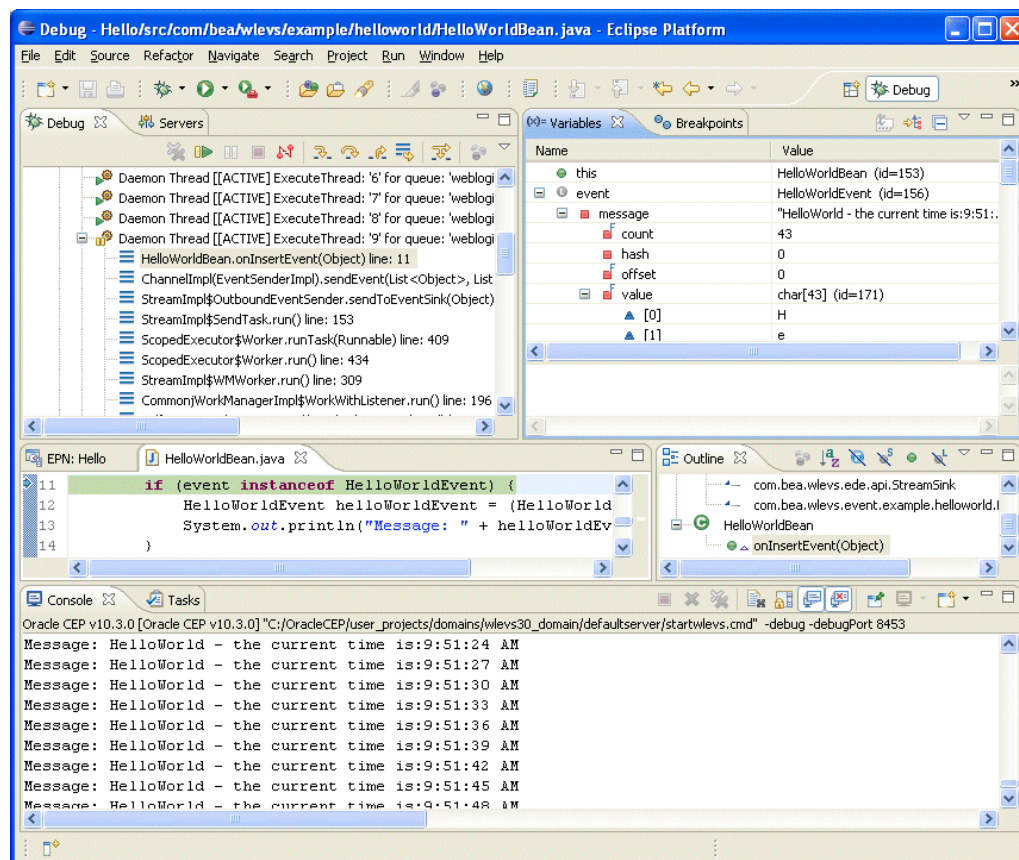
Figure J-13 Java Editor for HelloWorldBean



Right-click in the gutter on line 10 (to the left of the line number), and select **Toggle Breakpoint**.

The next time this `StreamSink` receives a message, it will stop at line 10 where the breakpoint was set and enter the Debug view as [Figure J-14](#) shows.

Figure J-14 HelloWorldBean in the Debugger



In the Variables view, you can expand the event variable and view the values of its attributes.

J.7 Next Steps

Examine the Oracle CEP examples available in the Oracle CEP server installer.

For more information, see "Oracle CEP Samples" in the *Oracle Complex Event Processing Getting Started*

Symbols

@Activate, I-2
@Prepare, I-4
@Resource, 1-22
@Rollback, I-5
@Service, I-6

A

accept-backlog, D-11
action, G-11, H-8
ActiveActiveGroupBean
 about, 20-7, 22-4
 configuring
 about, 23-8
 with High Availability, 23-11
 without High Availability, 23-9
 groupPattern
 default, 22-4, 23-17
 defining, 23-17
 notification group name default, 22-4, 23-17
 scalability, 22-4, 22-6
adapters
 cache access, 12-33
 config.xml, D-11
 csvgen, 25-4
 custom
 about, 14-1
 accessing configuration, 19-9
 adapter class as event sink, 14-8, 15-4
 adapter class as event source, 14-5
 adapter factories, 14-3, 14-11, 15-6
 bundle, 24-12
 configuring, 14-18, 15-8
 creating, 14-3, 14-4
 EPN assembly file, 14-16, 14-17, 15-6, 15-7
 event sinks, 14-2
 event sources, 14-2, 16-1
 extending, 19-1, 19-2, 19-4
 passing login credentials, 14-12
 Spring bean event sinks, 16-2
 Spring bean event sources, 16-1
 XSD creation, 19-6
EPN, 1-2
high availability

broadcast output adapter, 20-6
buffering output adapter, 20-6
correlating output adapter, 20-7
input adapter, 20-6
HTTP pub-sub server
 about, 8-1
 built-in adapters, 8-1
 clients, 8-2
 configuring, 8-6
 conversion to and from event types, 8-8
 conversion to and from JSON messages, 8-5
 custom adapters, 8-1
 custom converter, 8-8
 EPN assembly file, 8-10
 local publishing, 8-2
 remote publishing, 8-3
 security, 8-2
 subscribing, 8-4
 using, 8-5
icon, 6-18
JMS
 about, 7-1
 clients, 7-7
 conversion to and from event types, 7-2
 IIOP client, 7-7
 inbound, 7-2
 Oracle WebLogic JMS provider, 7-7
 Oracle WebLogic Server JMS provider, 7-7
 outbound, 7-3
 partitioning, 22-4, 22-6
 providers, 7-1
 scalability, 22-4, 22-6
 T3 client, 7-7
 Tibco EMS JMS provider, 7-10
amount, D-12
annotations
 @Activate, I-2
 @Prepare, I-4
 @Resource, 1-22
 @Rollback, I-5
 @Service, I-6
 about, I-1
 lifecycle, I-1
 OSGi service reference, I-1
 resource access, I-2
anyinteract-tolerance, 18-2

- API, 1-17
- application, D-13
- application context, 18-1
 - Oracle JDBC data cartridge, 18-3
 - Oracle Spatial, 18-1
- application library
 - about, 24-3
 - bundles, 4-29
 - creating, 24-5
 - defining, 24-14
 - deployment order, 24-5
 - development time dependency definition, 4-29
 - directories, 24-4
 - library directory, 24-4
 - library extensions directory, 24-4
 - managing, 24-13
 - order of deployment, 24-5
 - OSGi bundles, 4-29
 - updating, 24-30
- applications
 - assembly
 - about, 1-14, 24-1, 24-6
 - foreign stages, 24-11
 - manual, 24-7
 - Oracle CEP IDE for Eclipse, 24-6
 - configuration history management, 24-6
 - contents, 24-1
 - creating, 1-19
 - data cartridges, 1-14
 - debugging, 5-32
 - dependencies
 - about, 24-2
 - bootclasspath, 24-3
 - native code, 24-3
 - private, 24-2
 - shared, 24-2
 - deploying, 5-23
 - deployment order, 24-5
 - extending the EPN, 1-14
 - high availability, 1-14
 - libraries
 - about, 24-3
 - creating, 24-5
 - lifecycle
 - about, 1-15
 - event sinks, 1-17
 - event sources, 1-17
 - installing, 1-16
 - resume, 1-17
 - server startup, 1-16
 - StreamSender, 1-17
 - suspend, 1-16
 - uninstall, 1-17
 - update, 1-17
 - scalability, 1-14
 - Web services, 1-14
- application-timestamped channels, 9-4, C-5
- assembling
 - about, 1-14, 24-1, 24-6
 - foreign stages, 24-11

- manual, 24-7
- Oracle CEP IDE for Eclipse, 24-6
- assembly files, 1-11
- auth-constraint, F-4
- average-interval, D-13
- average-latency, D-14

B

- badging on EPN, 6-7
- BatchRelationSender
 - sendEvents, 1-8
- BatchRelationSink
 - about, 1-8
 - onEvents, 1-8
- batch-size, D-15
- BatchStreamSender
 - sendInsertEvents, 1-7
- BatchStreamSink
 - about, 1-8
 - onInsertEvents, 1-8
- batch-timeout, D-16
- bdb-config, F-5
- Berkeley DB persistent event store, 13-2
- binding, D-16
- bindings (jms-adapter), D-17
- bindings (processor), D-18
- blocking queue channels, 9-9, 9-12, 9-15, C-13, D-49
- bootclasspath, 24-3
- broadcast output adapter, 20-6
- buffering output adapter, 20-6
- buffer-size, D-19
- buffer-write-attempts, D-20
- buffer-write-timeout, D-21
- bundler.sh, 24-18
- bundles, 4-29

C

- cache
 - config.xml, D-21
 - icon, 6-19
- cache system
 - about, 12-1, D-22, D-25
 - Oracle CEP local cache, 12-1
 - Oracle Coherence, 12-1
 - third-party, 12-2
- caches
 - about, 12-1, D-21
 - accessing
 - adapter, 12-33
 - EPL statement, 12-31
 - EPL user-defined function, 12-35
 - errors, 12-29
 - JMX, 12-36
 - Oracle CQL statement, 12-28, 12-29
 - Oracle CQL user-defined function, 12-34
 - POJO, 12-34
- API, 12-5
- cache loader
 - Oracle CEP local cache, 12-13

- Oracle Coherence, 12-22
- cache store
 - Oracle CEP local cache, 12-14
 - Oracle Coherence, 12-22
- cache system
 - about, 12-1, D-22, D-25
 - Oracle CEP local cache, 12-1
 - Oracle Coherence, 12-1
 - third-party, 12-2
- configuring
 - Oracle CEP local cache, 12-6
 - Oracle Coherence, 12-14
 - third-party, 12-25
- EPL processor cache source, 11-5, 12-31
- EPN, 1-3, 1-4
- multi-server domains, 12-5
- Oracle CQL processor cache source, 10-11, 12-28
- caching-system, D-22
- canvas
 - moving, 6-11
- cause, G-10, H-7
- channel
 - http-pub-sub element, D-24
 - icon, 6-18
 - top-level element, D-23
- channel-constraints, F-7
- channel-resource-collection, F-7
- channels, F-6
 - about, 9-1, D-23
 - application-timestamped, 9-4
 - batching, 9-6
 - blocking queue, 9-9, 9-12, 9-15, C-13, D-49
 - component configuration file, 9-16
 - configuring, 9-6
 - EPN, 1-3
 - EPN assembly file, 9-17
 - event identity, 1-8, 9-3, C-13
 - EventPartitioner, 9-6
 - partitioning, 9-6
 - pass-through, 9-9, 9-12, 9-15, C-13, D-49
 - primary-key, 1-8, 9-3, C-13
 - query selector, 9-4, 9-5, 9-9, 9-13, 9-15, 10-3, D-67
 - relations, 1-7, 9-3
 - streams, 1-7, 9-3
 - system-timestamped, 9-4
 - when to use, 9-2
- cluster, F-9
- coherence-cache-config, D-24
- coherence-caching-system, D-25
- coherence-cluster-config, D-25
- collect-interval, D-26
- component configuration files
 - about, 1-12
 - accessing, 1-13
 - creating, 4-8
 - property placeholders, 1-13
- concurrent-consumers, D-27
- configuration badging on EPN, 6-7
- configuration files
 - accessing, 1-13

- assembly, 1-11
- component, 1-12
- EPN assembly, 1-11
- property placeholders, 1-13
- configuration history management
 - about, 24-6
- ConfigurationPropertyPlaceholderConfigurer, 1-13
- connection-jndi-name, D-27
- connection-password, D-28, D-29
- connection-pool-params, F-10
- connection-user, D-29
- context menus, 6-14
- correlating output adapter, 20-7
- cql, F-13
- csvgen adapter
 - about, 25-1
 - configuration, 25-4
 - data types, 2-5
- custom adapters
 - about, 14-1
 - accessing configuration, 19-9
 - adapter class as event sink, 14-8, 15-4
 - adapter class as event source, 14-5
 - adapter factories
 - about, 14-3
 - programming, 14-11, 15-6
 - bundle, 24-12
 - configuring
 - manually, 14-18, 15-8
 - creating
 - Ant, 14-3
 - manually, 14-4
 - EPN assembly file
 - about, 14-16, 15-6
 - declaring adapter, 14-17, 15-7
 - registering adapter, 14-16, 15-6
 - event sinks, 14-2
 - event sources, 14-2, 16-1
 - extending, 19-2, 19-4
 - about, 19-1
 - annotations, 19-2
 - XSD, 19-2
 - passing login credentials, 14-12
 - Spring bean event sinks, 16-2
 - Spring bean event sources, 16-1
 - XSD creation, 19-6
- custom persistent event store, 13-2

D

- data cartridges
 - about, 18-1
 - application context, 18-1
 - Oracle JDBC data cartridge
 - application context, 18-3
 - Oracle Spatial
 - application context, 18-1
- data types
 - csvgen adapter event type, 2-5
 - event types, 2-3

- Java class event type, 2-3
- JavaBean event type, 2-3
- java.util.Map event type, 2-4
- JDBC, 2-5
- table source event type, 2-5
- tuple event types, 2-4
- database, D-30
- dataset-name, D-31
- data-source, F-14
- data-source-params, F-15
- debug, F-18
- debugging
 - applications, 5-32
 - servers, 5-32
- default event store, 13-2
- delivery-mode, D-31
- deploying
 - about, 24-1
 - deploying applications, 24-38
 - order, 24-5
- deployment.xsd, B-3, E-1
- destination-jndi-name, D-32
- destination-name, D-32
- diagnostic-profiles, D-33
- direction, D-33
- domain, F-18
- driver-params, F-17
- duration, D-34

E

- enabled, D-35
- encrypted-password, D-36
- end, D-36
- end-location, D-37
- EPL processors
 - about, 11-1
 - cache source, 11-5, 12-31
 - component configuration file, 11-6
 - configuring, 11-3
 - EPN assembly file, 11-6
- EPN
 - about, 1-2
 - adapters, 1-2
 - assembly file
 - about, 1-11
 - creating, 4-6
 - caches, 1-3, 1-4
 - channels, 1-3
 - component configuration file
 - about, 1-12
 - creating, 4-8
 - components, 1-2
 - event beans, 1-3
 - extending, 1-14
 - nodes, 1-2
 - POJO, 1-3
 - processors, 1-3
 - Spring beans, 1-3
 - stages, 1-2

- EPN editor
 - about, 6-1
 - browsing types, 6-15
 - configuration badging, 6-7
 - connecting nodes, 6-28
 - context menus, 6-14
 - creating adapter nodes, 6-21
 - creating basic nodes, 6-19
 - creating nodes, 6-18
 - creating processor nodes, 6-26
 - deleting nodes, 6-30
 - event type repository editor, 6-10
 - filtering, 6-5
 - hyperlinking, configuration files, 6-12
 - hyperlinking, Oracle CQL statements, 6-13
 - laying out nodes, 6-30
 - layout, 6-5
 - link specification, 6-8
 - moving the canvas, 6-11
 - navigating, 6-11
 - nested elements, 6-9
 - opening, 6-1
 - printing, 6-7
 - renaming nodes, 6-30
 - shortcuts to configuration files, 6-11, 6-12, 6-13, 6-14
 - type browser, 6-15
 - using, 6-18
 - zooming, 6-5
- event beans
 - about, 15-1
 - accessing configuration, 19-9
 - bundle, 24-12
 - configuring, 15-7
 - creating, 15-2, 16-2
 - EPN, 1-3
 - EPN assembly file
 - about, 14-16, 15-6
 - declaring event bean, 14-17, 15-7
 - registering event bean factory, 14-16, 15-6
 - event bean class as event sink, 14-8, 15-4
 - event bean class as event source, 15-3
 - event sinks, 15-1, 15-2, 16-1
 - event sources, 14-2, 15-1, 16-1
 - extending, 19-2, 19-4
 - about, 19-1
 - annotations, 19-2
 - XSD, 19-2
 - factories
 - about, 15-2
 - programming, 14-11, 15-6
 - icon, 6-19
 - passing login credentials, 14-12
 - XSD creation, 19-6
- event identity, 1-8, C-13
- Event Inspector service
 - about, 26-1
 - clients, 26-4
 - event types, 26-2
 - HTTP pub-sub

- channel, 26-3
- configuration, 26-4
- local server configuration, 26-5
- remote server configuration, 26-5
- server, 26-4
- injecting events, 26-2
- JSON format
 - about, 26-2
 - binding, 26-3
 - event-type, 26-3
 - operation, 26-3
 - value, 26-3
- tracing events, 26-1
- event partitioner channel
 - about, 22-2
 - configuring
 - about, 23-1
 - custom, 23-4
 - default, 23-2
 - custom implementation, 22-3
 - default implementation, 22-2
 - initialization, 22-3
 - load balancing, 22-3
 - restrictions, 22-3
 - threading, 22-3
- Event Processing Network. *See* EPN
- event record and playback
 - about, 13-1
 - configuring
 - application, 13-4
 - custom event store provider, 13-12
 - event playback, 13-8
 - event recording, 13-5
 - event store, 13-5
 - database schema, 13-11
 - event store
 - about, 13-2
 - Berkeley DB, 13-2
 - custom, 13-2
 - default, 13-2
 - playing events, 13-3
 - querying stored events, 13-3
 - recording events, 13-3
 - starting and stopping, 13-10
- event sources
 - pull, 1-4, 12-29, 12-31
 - push, 6-28, 9-2, 12-31
- event store
 - about, 13-2
 - Berkeley DB, 13-2
 - configuring, 13-5
 - custom, 13-2
 - custom provider, 13-12
 - default, 13-2
- event type repository editor
 - about, 6-10
 - JavaBean event types, 2-6
 - tuple event types, 2-12
- event types
 - about, 1-6, 2-1

- accessing the EventTypeRepository, 2-22
- conversion to and from HTTP pub-sub server
 - messages, 8-8
- conversion to and from JMS messages, 7-2
- conversion to and from JSON messages, 8-5
- creating, 2-5
- data types, 2-3
- Event Inspector service events, 26-2
- event type builder factory, 2-21
- event type repository editor, 6-10
- EventTypeRepository, 2-2
- immutable, 2-2
- instantiation, 2-2
- Java Class, 2-2
- JavaBean, 2-2
- java.util.Map, 2-2
- serialization, 2-3
- sharing between bundles, 2-23
- tuple, 2-2
- event-bean, D-38
- EventPartitioner. *See* event partitioner channel
- EventTypeRepository, 2-2
- event-store, F-19
- event-type, D-39
- event-type-list, D-39
- eviction-policy, D-40
- exported-jndi-context, F-20
- exporting EPN image, 6-7
- exporting image, 6-7
- external relation
 - must be joined with s[now], 12-29
 - querying, 12-29

F

- failover, 20-3
- filtering EPN, 6-5
- foreign stages
 - about, 1-5
 - application assembly, 24-11
 - classpath, 24-11
 - dependencies, 24-11
 - on EPN, 1-5
 - restrictions, 1-6

G

- group-binding, D-41
- groupPattern, 23-17

H

- heartbeat, 9-9, 9-15, 21-27, D-42
 - application-timestamped channels, 9-4
 - configuring high availability input adapter, 21-27
 - configuring system-timestamped channel, 9-9, 9-15
 - StreamSender API, 1-7
 - system timestamped relations, D-42
 - system timestamped streams, D-42
 - system-timestamped channels, 9-4

- high availability
 - application design patterns, 20-13
 - application type
 - Type 1, 20-14
 - Type 2, 20-15
 - components
 - ActiveActiveGroupBean, 20-7
 - broadcast output adapter, 20-6
 - buffering output adapter, 20-6
 - correlating output adapter, 20-7
 - input adapter, 20-6
 - notification groups bean, 20-7
 - lifecycle, 20-2
 - failover, 20-3
 - primary failure, 20-3
 - rejoining the multi-server domain, 20-3, 20-14
 - secondary failure, 20-3
 - server restart, 20-3, 20-14
 - Oracle CQL query restrictions, 20-17
 - performance tuning
 - broadcast adapter, 27-2
 - host configuration, 27-2
 - input adapter, 27-2
 - Oracle Coherence, 27-3
 - quality of service, 27-2
 - quality of service, 20-9
 - light-weight queue trimming, 20-10
 - precise recovery with JMS, 20-11
 - simple failover, 20-9
 - simple failover with buffering, 20-10
 - scalability, 20-8
 - use cases, 20-12
 - warm-up-window time, 20-14
- HTTP pub-sub server adapters
 - about, 8-1
 - built-in adapters, 8-1
 - clients, 8-2
 - configuring, 8-6
 - conversion to and from event types, 8-8
 - conversion to and from JSON messages, 8-5
 - custom adapters, 8-1
 - custom converter, 8-8
 - EPN assembly file, 8-10
 - local publishing, 8-2
 - remote publishing, 8-3
 - security, 8-2
 - subscribing, 8-4
 - using, 8-5
- http-pubsub, F-20
- http-pub-sub-adapter, D-42
- hyperlinking
 - configuration files, 6-12
 - Oracle CQL statements, 6-13

I

- I10n_msgcat.dtd, G-1, H-1
- icon
 - adapter, 6-18
 - cache, 6-19

- channel, 6-18
- event bean, 6-19
- processor, 6-18
- Spring bean, 6-19
- table, 6-19
- idle-time, D-43
- injection
 - dynamic, 1-24
 - static
 - about, 1-22
 - dynamic resource names, 1-23
 - static resource names, 1-22
- input adapter, 20-6
- is-relation, C-12
- is-total-order, 9-11, C-5

J

- JAR files, 4-29
- Java Database Connectivity. *See* JDBC
- Java Message Service. *See* JMS
- Java Naming and Directory Interface. *See* JNDI
- jc:jdbc-ctx, 18-3
- JDBC
 - configuring
 - application library, 24-18, 24-21
 - OSGi bundle, 24-18, 24-21
 - data cartridge, 18-3
 - event types, 2-5
- jdbc:jdbc-context, 18-3
- jetty, F-21
- jetty-web-app, F-22
- JMS
 - partitioning, 22-4, 22-6
- JMS adapters
 - about, 7-1
 - clients, 7-7
 - conversion to and from event types, 7-2
 - IIOP client, 7-7
 - inbound, 7-2
 - Oracle WebLogic JMS provider, 7-7
 - outbound, 7-3
 - providers
 - about, 7-1
 - Oracle WebLogic Server, 7-7
 - Tibco EMS, 7-10
 - scalability, 22-4, 22-6
 - T3 client, 7-7
 - Tibco EMS JMS provider, 7-10
- jms-adapter, D-44, D-45
- jmx, F-23
- JMX cache access, 12-36
- JNDI resource access, 1-24
- jndi-context, F-24
- jndi-provider-url, D-46
- JSON messages, 8-5

L

- layout of EPN, 6-5
- library management, 4-29

- link specification on EPN, 6-8
- listeners, D-46
- load generator
 - about, 25-1
 - configuring and running, 25-1
 - csvgen adapter, 25-4
 - data feed file, 25-3
 - property file, 25-2
- local servers, 5-3
- locale_message_catalog, H-2
- location, D-47
- log-file, F-24
- logmessage, G-5, H-3
- log-stdout, F-26

M

- max-latency, D-48
- max-size, D-49
- max-threads, D-49
- message, G-7, H-5
- message_catalog, G-2
- messagebody, G-8, H-5
- messagedetail, G-10, H-6
- message-filter-class, F-28
- message-filter-name, F-28
- message-filters, F-28
- message-selector, D-50
- msgcat.dtd, G-1

N

- name, D-50, F-29
- native code application dependencies, 24-3
- nested elements on EPN, 1-4, 6-9
- nested stages, 1-4
- netio, D-51, F-29
- netio-client, F-30
- nodes
 - adapter, 6-18
 - cache, 6-19
 - channel, 6-18
 - event bean, 6-19
 - processor, 6-18
 - Spring bean, 6-19
 - table, 6-19
- non-class file management, 4-29
- notification groups bean, 20-7
- num-threads, D-51

O

- ocep-jdbc.xsd, 18-4
- ocep-spatial.xsd, 18-2
- onDeleteEvent, 1-8
- onEvents, 1-8
- onInsertEvent, 1-8
- onInsertEvents, 1-8
- onUpdateEvent, 1-8
- Oracle CEP IDE for Eclipse, 1-19
 - about, 3-1

- configuring
 - memory requirements, 3-11
 - preferences, 4-43
- EPN editor
 - about, 6-1
 - browsing types, 6-15
 - configuration badging, 6-7
 - connecting nodes, 6-28
 - context menus, 6-14
 - creating basic nodes, 6-19
 - creating nodes, 6-18
 - creating processor nodes, 6-26
 - deleting nodes, 6-30
 - event type repository editor, 6-10
 - exporting image, 6-7
 - filtering, 6-5
 - hyperlinking, configuration files, 6-12
 - hyperlinking, Oracle CQL statements, 6-13
 - laying out nodes, 6-30
 - layout, 6-5
 - link specification, 6-8
 - moving the canvas, 6-11
 - navigating, 6-11
 - nested elements, 6-9
 - opening, 6-1
 - printing, 6-7
 - renaming nodes, 6-30
 - shortcuts to configuration files, 6-11, 6-12, 6-13, 6-14
 - using, 6-18
 - zooming, 6-5
- installing
 - from Oracle CEP, 3-7
 - latest, 3-2
- memory requirements, 3-11
- preferences, 4-43
- projects
 - about, 4-1
 - component configuration files, 4-8
 - creating, 4-2
 - EPN assembly files, 4-6
 - exporting, 4-10
 - managing libraries, 4-29
 - managing non-class files, 4-29
 - upgrading, 4-13
- servers
 - about, 5-1
 - attached, 5-2
 - attaching, 5-22
 - attaching, to local, 5-21
 - attaching, to remote, 5-22
 - configuring, 5-26, 5-28
 - connection and control settings, 5-26
 - creating, 5-3
 - debugging, 5-32
 - deploying, 5-23
 - domain settings, 5-28
 - local, 5-3
 - managed, 5-2
 - managing, 5-19

- Oracle CEP Visualizer, 5-30
 - remote, 5-10
 - runtime, 5-16
 - starting, 5-19
 - stopping, 5-20
 - tutorial, J-1
 - Oracle CEP local cache
 - configuring, 12-6
 - cache loader, 12-13
 - cache store, 12-14
 - event listener, 12-11
 - event source, 12-13
 - Oracle CEP Visualizer
 - about, 1-19
 - starting, 5-30
 - Oracle Coherence
 - caches, 12-1
 - configuring, 12-14
 - cache loader, 12-22
 - cache store, 12-22
 - caches, 12-17
 - caching system, 12-17
 - event listener, 12-20
 - event source, 12-22
 - Oracle CQL processors
 - about, 10-1
 - cache source, 10-11, 12-28
 - component configuration file, 10-11
 - configuring, 10-3
 - EPN assembly file, 10-12
 - query selector, 10-3
 - table source, 10-7
 - Oracle Fusion Middleware Home, 5-6, 5-12, 5-18
 - Oracle JDBC data cartridge
 - application context
 - about, 18-3
 - Oracle Middleware Home, 5-6, 5-12, 5-18
 - Oracle Spatial
 - application context
 - about, 18-1
 - anyinteract-tolerance, 18-2
 - reciprocal of flatening, 18-3
 - rof, 18-3
 - semi-major axis, 18-3
 - sma, 18-3
 - srid, 18-3
 - tolerance, 18-3
 - Oracle WebLogic JMS provider, 7-7
 - OSGi, 1-1
 - about, A-1
 - application library, 4-29
 - bundles, 4-29
- P**
-
- param, D-52
 - parameter, D-53
 - params, D-53
 - partitioning
 - channel, 22-2
 - JMS topic, 22-4, 22-6
 - pass-through channels, 9-9, 9-12, 9-15, C-13, D-49
 - password, D-54
 - path, F-30
 - performance tuning
 - about, 27-1
 - EPN
 - ActiveActiveGroupBean, 27-1
 - batching channel, 27-1
 - event partitioner channel, 27-1
 - high availability
 - broadcast adapter, 27-2
 - host configuration, 27-2
 - input adapter, 27-2
 - Oracle Coherence, 27-3
 - quality of service, 27-2
 - playback-parameters, D-55
 - playback-speed, D-56
 - POJO
 - event beans, 1-3
 - Spring Bean, 1-3
 - primary-key, 1-8, 9-3, C-13
 - printing, 6-7
 - private application dependencies, 24-2
 - processor (EPL), D-57
 - processor (Oracle CQL), D-58
 - processors
 - about, 1-3
 - creating in EPN, 6-26
 - EPL
 - about, 11-1
 - cache source, 11-5, 12-31
 - component configuration file, 11-6
 - configuring, 11-3
 - EPN assembly file, 11-6
 - icon, 6-18
 - Oracle CQL
 - about, 10-1
 - cache source, 10-11, 12-28
 - component configuration file, 10-11
 - configuring, 10-3
 - EPN assembly file, 10-12
 - query selector, 10-3
 - table source, 10-7
 - profile, D-59
 - programming model
 - about, 1-1
 - accessing component configuration files, 1-13
 - API, 1-17
 - application assembly, 1-14
 - application lifecycle
 - about, 1-15
 - event sinks, 1-17
 - event sources, 1-17
 - installing, 1-16
 - resume, 1-17
 - server startup, 1-16
 - StreamSender, 1-17
 - suspend, 1-16
 - uninstall, 1-17

- update, 1-17
- assembly files, 1-11
- component configuration files, 1-12
- component configuration property placeholders, 1-13
- creating applications, 1-19
- creating component configuration file, 4-8
- creating EPN assembly file, 4-6
- data cartridges, 1-14
- EPN, 1-2
- EPN assembly files, 1-11
- event consumers, 1-6
- event emitters, 1-6
- event types
 - about, 1-6, 2-1
 - accessing the EventTypeRepository, 2-22
 - conversion to and from HTTP pub-sub server messages, 8-8
 - conversion to and from JMS messages, 7-2
 - conversion to and from JSON messages, 8-5
 - creating, 2-5
 - data types, 2-3
 - event type builder factory, 2-21
 - event type repository editor, 6-10
 - EventRepository, 2-2
 - immutable, 2-2
 - instantiation, 2-2
 - Java Class, 2-2
 - JavaBean, 2-2
 - java.util.Map, 2-2
 - serialization, 2-3
 - sharing between bundles, 2-23
 - tuple, 2-2
- extending the EPN, 1-14
- high availability, 1-14
- high availability lifecycle, 20-2
 - failover, 20-3
 - primary failure, 20-3
 - rejoining the multi-server domain, 20-3, 20-14
 - secondary failure, 20-3
 - server restart, 20-3, 20-14
 - warm-up-window time, 20-4
- IDE, 1-19
- Oracle CEP Visualizer, 1-19
- OSGi, 1-1
- relation sinks
 - about, 1-8
- relation sources
 - about, 1-8
 - BatchRelationSender, 1-8
 - RelationSender, 1-8
- resource access
 - about, 1-21
 - dynamic resource injection, 1-24
 - JNDI, 1-24
 - resource name resolution, 1-25
 - static resource injection, 1-22
- scalability, 1-14
- stream sinks
 - about, 1-8

- BatchRelationSink, 1-8
- BatchStreamSink, 1-8
- RelationSink, 1-8
- StreamSink, 1-8
- stream sources
 - about, 1-7
 - BatchStreamSender, 1-7
 - StreamSender, 1-7
- Web services, 1-14
- projects
 - about, 4-1
 - component configuration files, 4-8
 - creating, 4-2
 - EPN assembly files, 4-6
 - exporting, 4-10
 - managing libraries, 4-29
 - managing non-class files, 4-29
 - upgrading, 4-13
- property files, 4-29
- property placeholders, 1-13
- provider-name, D-60
- pubsub-bean, F-31
- pull event sources, 1-4, 12-29, 12-31
- push event sources, 6-28, 9-2, 12-31

Q

- query, D-61

R

- rdbms-event-store-provider, F-32
- record-parameters, D-62
- relations, 1-7
 - channels, 9-3
 - heartbeat, D-42
- RelationSender
 - about, 1-8
 - event identify, 1-8
 - sendDeleteEvent, 1-8
 - sendUpdateEvent, 1-8
- RelationSink
 - about, 1-8
 - onDeleteEvent, 1-8
 - onUpdateEvent, 1-8
- remote servers, 5-10
- repeat, D-62
- resource access
 - dynamic resource injection, 1-24
 - JNDI, 1-24
 - resource name resolution, 1-25
 - static resource injection, 1-22
 - dynamic resource names, 1-23
 - static resource names, 1-22
- resource name resolution, 1-25
- rmi, F-33
- rof, 18-3
- rule, D-63
- rules, D-64

S

- scalability
 - components
 - about, 22-2
 - ActiveActiveGroupBean, 22-4, 22-6, 23-8, 23-9, 23-11
 - event partitioner channel, 22-2, 23-1, 23-2, 23-4
 - high availability, 22-1
 - JMS, 22-4, 22-6
- scheduler, F-34
- schedule-time-range, D-65
- schedule-time-range-offset, D-65
- schema. *See* XSD
- selector, D-66
- sendDeleteEvent, 1-8
- sendEvents, 1-8
- sendHeartbeat, 1-7
- sendInsertEvent, 1-7
- sendInsertEvents, 1-7
- sendUpdateEvent, 1-8
- server-config, F-35
- server-context-path, D-68
- servers
 - about, 5-1
 - attached, 5-2
 - attaching, 5-22
 - attaching, to local, 5-21
 - attaching, to remote, 5-22
 - configuring, 5-26, 5-28
 - connection and control settings, 5-26
 - creating, 5-3
 - debugging, 5-32
 - deploying, 5-23
 - domain settings, 5-28
 - local, 5-3
 - managed, 5-2
 - managing, 5-19
 - Oracle CEP Visualizer, 5-30
 - remote, 5-10
 - runtime, 5-16
 - starting, 5-19
 - stopping, 5-20
- server-url, D-68
- services, F-37
- session-ack-mode-name, D-69
- session-transacted, D-70
- shared application dependencies, 24-2
- show-detail-error-message, F-38
- sliding window output adapter. *See* buffering output adapter
- sma, 18-3
- spatial:context, 18-1
- Spring bean icon, 6-19
- Spring beans
 - about, 16-1
 - EPN, 1-3
 - EPN assembly file
 - about, 16-5
 - declaring Spring bean, 16-5
 - Spring bean class as event sink, 16-4

- Spring bean class as event source, 16-3
- spring-wlevs-v11_1_1_3.xsd, B-1, C-1
- srid, 18-3
- ssl, F-39
- stage, D-70
- start, D-71
- start-location, D-72
- start-stage, D-73
- store-policy-parameters, D-74
- stream, D-74
- streams, 1-7
 - channels, 9-3
 - heartbeat, D-42
 - query selector, 9-4, 9-5, 9-9, 9-13, 9-15, 10-3, D-67
- StreamSender
 - about, 1-7
 - sendHeartbeat, 1-7
 - sendInsertEvent, 1-7
- StreamSink
 - about, 1-8
 - onInsertEvent, 1-8
- symbol, D-75
- symbols, D-75
- system-timestamped channels, 9-4, C-5

T

- table
 - icon, 6-19
 - source, 10-7
- testing
 - Event Inspector service
 - about, 26-1
 - clients, 26-4
 - event types, 26-2
 - HTTP pub-sub channel, 26-3
 - injecting events, 26-2
 - tracing events, 26-1
 - load generator
 - about, 25-1
 - configuring and running, 25-1
 - csvgen adapter, 25-4
 - data feed file, 25-3
 - property file, 25-2
- third-party cache configuration, 12-25
- threshold, D-76
- throughput, D-77
- throughput-interval, D-78
- Tibco EMS JMS provider, 7-10
- timeout-seconds, F-40
- time-range, D-79
- time-range-offset, D-79
- timestamps
 - application, C-5
 - system, C-5
- time-to-live, D-80
- tolerance, 18-3
- tools
 - Oracle CEP IDE for Eclipse, 1-19, 3-1
 - Oracle CEP Visualizer, 1-19

transaction-manager, F-41
tuple
 data types, 2-4
tutorial, J-1
Type 1 high availability applications, 20-14
Type 2 high availability applications, 20-15
type browser, 6-15

U

unit, D-81
user, D-82
use-secure-connections, F-43

V

value, D-82
view, D-83

W

warm-up-window time, 20-4, 20-14
Web Services
 about, 17-1
 exposing an Oracle CEP application as, 17-2
 invoking from an Oracle CEP application, 17-1
weblogic-instances, F-44
weblogic-jta-gateway, F-45
weblogic-rmi-client, F-46
wlevs:adapter, C-3
wlevs:application-timestamped, C-5
wlevs:cache, C-6
wlevs:cache-listener, C-7
wlevs:cache-source, C-9
wlevs:channel, C-11
wlevs:deployment, E-2
wlevs:event-bean, C-13
wlevs:event-type, C-16
wlevs:event-type-repository, C-15
wlevs:expression, C-17
wlevs:factory, C-17
wlevs:function, C-18
wlevs:instance-property, C-25
wlevs:listener, C-26
wlevs:loader, C-8
wlevs:metadata, C-27
wlevs:processor, C-28
wlevs:property, C-29
wlevs:source, C-30
wlevs:table-source, C-31
wlevs_application_config.xsd, B-2, D-1
wlevs_server_config.xsd, B-3, F-1
work-manager, D-84, F-27, F-46
work-manager-name, D-84
write-behind, D-85
write-none, D-86
write-through, D-86

X

xa-params, F-47

XSD

about, B-1
component configuration file, B-2, D-1
config.xml, B-3
deployment.xsd, B-3, E-1
 wlevs:deployment, E-2
EPN assembly file, B-1, C-1, E-1, F-1
I10n_msgcat.dtd, G-1, H-1
 action, H-8
 cause, H-7
 logmessage, H-3
 message, H-5
 messagebody, H-5
 messagedetail, H-6
i10n_msgcat.dtd
 locale_message_catalog, H-2
locale message catalog, H-1
message catalog, G-1
msgcat.dtd, G-1
 action, G-11
 cause, G-10
 logmessage, G-5
 message, G-7
 message_catalog, G-2
 messagebody, G-8
 messagedetail, G-10
ocep-jdbc.xsd, 18-4
ocep-spatial.xsd, 18-2
server configuration file, B-3
spring-wlevs-v11_1_1_3.xsd, B-1, C-1
 wlevs:adapter, C-3
 wlevs:application-timestamped, C-5
 wlevs:cache, C-6
 wlevs:cache-listener, C-7
 wlevs:cache-source, C-9
 wlevs:cache-store, C-10
 wlevs:caching-system, C-10
 wlevs:channel, C-11
 wlevs:event-bean, C-13
 wlevs:event-type, C-16
 wlevs:event-type-repository, C-15
 wlevs:expression, C-17
 wlevs:factory, C-17
 wlevs:function, C-18
 wlevs:instance-property, C-25
 wlevs:listener, C-26
 wlevs:loader, C-8
 wlevs:metadata, C-27
 wlevs:processor, C-28
 wlevs:property, C-29
 wlevs:source, C-30
 wlevs:table-source, C-31
wlevs_application_config.xsd, B-2, D-1
 accept-backlog, D-11
 adapter, D-11
 amount, D-12
 application, D-13
 average-interval, D-13
 average-latency, D-14
 batch-size, D-15

- batch-timeout, D-16
- binding, D-16
- bindings (jms-adapter), D-17
- bindings (processor), D-18
- buffer-size, D-19
- buffer-write-attempts, D-20
- buffer-write-timeout, D-21
- cache, D-21
- caching-system, D-22
- channel (http-pub-sub element), D-24
- channel (top-level element), D-23
- coherence-cache-config, D-24
- coherence-caching-system, D-25
- coherence-cluster-config, D-25
- collect-interval, D-26
- concurrent-consumers, D-27
- connection-jndi-name, D-27
- connection-password, D-28, D-29
- connection-user, D-29
- database, D-30
- dataset-name, D-31
- delivery-mode, D-31
- destination-jndi-name, D-32
- destination-name, D-32
- diagnostic-profiles, D-33
- direction, D-33
- duration, D-34
- enabled, D-35
- encrypted-password, D-36
- end, D-36
- end-location, D-37
- event-bean, D-38
- event-type, D-39
- event-type-list, D-39
- eviction-policy, D-40
- group-binding, D-41
- heartbeat, D-42
- http-pub-sub-adapter, D-42
- idle-time, D-43
- jms-adapter, D-44, D-45
- jndi-provider-url, D-46
- listeners, D-46
- location, D-47
- max-latency, D-48
- max-size, D-49
- max-threads, D-49
- message-selector, D-50
- name, D-50
- netio, D-51
- num-threads, D-51
- param, D-52
- parameter, D-53
- params, D-53
- password, D-54
- playback-parameters, D-55
- playback-speed, D-56
- processor (EPL), D-57
- processor (Oracle CQL), D-58
- profile, D-59
- provider-name, D-60
- query, D-61
- record-parameters, D-62
- repeat, D-62
- rule, D-63
- rules, D-64
- schedule-time-range, D-65
- schedule-time-range-offset, D-65
- selector, D-66
- server-context-path, D-68
- server-url, D-68
- session-ack-mode-name, D-69
- session-transacted, D-70
- stage, D-70
- start, D-71
- start-location, D-72
- start-stage, D-73
- store-policy-parameters, D-74
- stream, D-74
- symbol, D-75
- symbols, D-75
- threshold, D-76
- throughput, D-77
- throughput-interval, D-78
- time-range, D-79
- time-range-offset, D-79
- time-to-live, D-80
- unit, D-81
- user, D-82
- value, D-82
- view, D-83
- work-manager, D-84
- work-manager-name, D-84
- write-behind, D-85
- write-none, D-86
- write-through, D-86
- wlevs_server_config.xsd, B-3, F-1
 - auth-constraint, F-4
 - bdb-config, F-5
 - channel-constraints, F-7
 - channel-resource-collection, F-7
 - channels, F-6
 - cluster, F-9
 - connection-pool-params, F-10
 - cql, F-13
 - data-source, F-14
 - data-source-params, F-15
 - debug, F-18
 - domain, F-18
 - driver-params, F-17
 - event-store, F-19
 - exported-jndi-context, F-20
 - http-pubsub, F-20
 - jetty, F-21
 - jetty-web-app, F-22
 - jmx, F-23
 - jndi-context, F-24
 - log-file, F-24
 - log-stdout, F-26
 - message-filter-class, F-28
 - message-filter-name, F-28

message-filters, F-28
name, F-29
netio, F-29
netio-client, F-30
path, F-30
pubsub-bean, F-31
rdbms-event-store-provider, F-32
rmi, F-33
scheduler, F-34
server-config, F-35
services, F-37
show-detail-error-message, F-38
ssl, F-39
timeout-seconds, F-40
transaction-manager, F-41
use-secure-connections, F-43
weblogic-instances, F-44
weblogic-jta-gateway, F-45
weblogic-rmi-client, F-46
work-manager, F-27, F-46
xa-params, F-47

Z

zooming EPN, 6-5

