

Oracle® CEP
Getting Started
Release 11gR1 (11.1.1)
E14476-02

October 2009

Oracle CEP Getting Started Release 11gR1 (11.1.1)

E14476-02

Copyright © 2007, 2009, 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	xiii
Audience	xiii
Documentation Accessibility	xiii
Related Documents	xiv
Conventions	xiv
1 Overview of Oracle CEP	
1.1 Introduction to Oracle Complex Event Processing.....	1-1
1.2 Conceptual Overview of Oracle CEP.....	1-1
1.3 Event Processing Networks.....	1-3
1.4 Use Cases.....	1-4
1.5 Summary of Oracle CEP Features	1-4
1.6 Supported Platforms.....	1-6
1.7 Oracle CEP IDE for Eclipse.....	1-6
1.8 Oracle CEP Visualizer	1-8
1.9 New Features in Oracle CEP Release 11gR1 (11.1.1)	1-8
1.9.1 Multi-Server Domains and Clustering	1-9
1.9.2 Caching.....	1-9
1.9.3 Record and Playback of Events	1-9
1.9.4 Visualizer Administration Console.....	1-9
1.9.5 HTTP Publish-Subscribe Adapter	1-10
1.9.6 JMS Adapter	1-10
1.9.7 Configuration Wizard for Creating Domains.....	1-10
1.9.8 Oracle Continuous Query Language (Oracle CQL)	1-10
1.9.9 Event Processing Language (EPL).....	1-12
1.9.10 Security.....	1-13
1.9.11 Event Beans and Spring Beans.....	1-13
1.9.12 Suspend and Resume Applications	1-14
1.9.13 Domain Directory Structure Changes	1-14
1.10 Next Steps	1-14
2 Installing Oracle CEP	
2.1 Before You Begin.....	2-1
2.2 Installation Overview	2-2
2.3 Installing Oracle CEP in Graphical Mode	2-2

2.4	Installing Oracle CEP in Console Mode	2-5
2.5	Installing Oracle CEP in Silent Mode.....	2-7
2.5.1	Creating a silent.xml File for Silent-Mode Installation	2-8
2.5.2	Guidelines for Component Selection	2-9
2.5.3	Returning Exit Codes to the Command Window	2-10
2.6	Installing an Oracle CEP Patch Using Smart Update	2-10
2.7	Post-Installation Steps	2-11
2.8	Upgrading to Oracle CEP Release 11gR1 (11.1.1).....	2-11
2.8.1	Upgrading a WebLogic Event Server 2.0 Domain to Oracle CEP 10.3	2-11
2.8.2	Upgrading an Oracle CEP 10.3 Domain to Oracle CEP Release 11gR1 (11.1.1).....	2-13
2.8.3	Upgrading a WebLogic Event Server 2.0 Application to Run on Oracle CEP 10.3.	2-14
2.8.4	Upgrading an Oracle CEP 10.3 Application to Run on Oracle CEP Release 11gR1 (11.1.1) 2-16	
2.8.5	Backward Compatibility Issues	2-18

3 Oracle CEP Samples

3.1	Overview of the Samples Provided in the Distribution Kit.....	3-1
3.1.1	Ready-to-Run Samples.....	3-2
3.1.2	Sample Source	3-2
3.2	Installing the Samples	3-3
3.3	Using Oracle CEP Visualizer With the Samples.....	3-3
3.4	Increasing the Performance of the Samples	3-3
3.5	Setting Your Development Environment.....	3-4
3.5.1	How to Set Your Development Environment on Windows.....	3-4
3.5.2	How to Set Your Development Environment on UNIX	3-5
3.6	HelloWorld Example.....	3-6
3.6.1	Running the HelloWorld Example from the helloworld Domain.....	3-7
3.6.2	Building and Deploying the HelloWorld Example from the Source Directory	3-8
3.6.3	Description of the Ant Targets to Build Hello World	3-9
3.6.4	Implementation of the HelloWorld Example	3-9
3.6.5	The HelloWorld EPN Assembly File	3-11
3.6.6	The HelloWorld Component Configuration File	3-13
3.7	Foreign Exchange (FX) Example.....	3-14
3.7.1	Running the Foreign Exchange Example	3-15
3.7.2	Building and Deploying the Foreign Exchange Example from the Source Directory.....	3-16
3.7.3	Description of the Ant Targets to Build FX.....	3-17
3.7.4	Implementation of the FX Example	3-17
3.7.5	The FX EPN Assembly File	3-19
3.7.6	The FX Processor Component Configuration Files	3-22
3.7.6.1	FX Processor Component Configuration File: spreader.xml	3-23
3.7.6.2	FX Processor Component Configuration File: SummarizeResults.xml.....	3-25
3.8	Signal Generation Example	3-26
3.8.1	Running the Signal Generation Example	3-27
3.8.2	Building and Deploying the Signal Generation Example from the Source Directory.....	3-29
3.8.3	Description of the Ant Targets to Build Signal Generation.....	3-30

3.8.4	Implementation of the Signal Generation Example	3-30
3.8.5	The Signal Generation EPN Assembly File.....	3-31
3.8.6	The Signal Generation Component Configuration Files.....	3-34
3.9	Event Record and Playback Example	3-36
3.9.1	Running the Event Record/Playback Example	3-37
3.9.2	Building and Deploying the Event Record/Playback Example from the Source Directory 3-44	
3.9.3	Description of the Ant Targets to Build the Record and Playback Example	3-45
3.9.4	Implementation of the Record and Playback Example.....	3-46
3.10	Oracle Continuous Query Language (CQL) Example.....	3-47
3.10.1	Running the CQL Example	3-48
3.10.2	Building and Deploying the CQL Example.....	3-49
3.10.3	Description of the Ant Targets to Build the CQL Example	3-50
3.10.4	Implementation of the CQL Example.....	3-50
3.10.4.1	Creating the Missing Event Query.....	3-50
3.10.4.2	Creating the Moving Average Query	3-75

Glossary

Index

List of Examples

1-1	MATCH_RECOGNIZE Query	1-11
2-1	Sample silent.xml File for Silent-Mode Installation	2-8
2-2	Sample Windows Command File Displaying Silent-Mode Exit Codes	2-10
2-3	Adapter Using loadgen Provider	2-16
2-4	Registering a StockTick Event	2-16
2-5	Spring-DM Declared Adapter Factory	2-18
2-6	wlevs:factory	2-18
3-1	HelloWorld EPN Assembly File	3-11
3-2	HelloWorld Component Configuration File	3-13
3-3	FX EPN Assembly File	3-19
3-4	Nested Component Definition	3-22
3-5	FX Processor Component Configuration File: spreader.xml	3-23
3-6	FX Processor Component Configuration File: SummarizeResults.xml	3-26
3-7	Signal Generation EPN Assembly File	3-32
3-8	Signal Generation Component Configuration File	3-34
3-9	replay Application Configuration File config.xml: adapter Element	3-41
3-10	replay Oracle CEP Server Configuration File config.xml: data-source and rdbms-event-store-provider Elements	3-41

List of Figures

1-1	Example Event -Driven System	1-2
1-2	Oracle CEP Application	1-3
1-3	Oracle CEP IDE for Eclipse.....	1-7
1-4	Oracle CEP Visualizer	1-8
3-1	The HelloWorld Example Event Processing Network.....	3-7
3-2	FX Example Event Processing Network.....	3-14
3-3	The Signal Generation Example Event Processing Network	3-26
3-4	Signal Generation Dashboard	3-29
3-5	The Event Record and Playback Example Event Processing Network.....	3-37
3-6	Oracle CEP Visualizer Logon Screen	3-39
3-7	Oracle CEP Visualizer Dashboard.....	3-40
3-8	Event Record Tab	3-41
3-9	Start Recording Alert Dialog.....	3-42
3-10	Event Playback Tab.....	3-43
3-11	Start Playback Alert Dialog	3-43
3-12	Stream Visualizer	3-44
3-13	The CQL Example Event Processing Network.....	3-47
3-14	Oracle CEP Visualizer Logon Screen	3-52
3-15	Oracle CEP Visualizer Dashboard.....	3-53
3-16	CQL Application Screen: General Tab.....	3-54
3-17	CQL Application: Event Processing Network Tab	3-55
3-18	Oracle CQL Processor: General Tab.....	3-56
3-19	Oracle CQL Processor: Query Wizard Tab	3-57
3-20	Template Tab	3-58
3-21	SSource Configuration Dialog.....	3-59
3-22	Pattern Configuration Dialog: Pattern Tab	3-60
3-23	Pattern Configuration Dialog: Define Tab	3-61
3-24	Expression Builder: CustOrder	3-62
3-25	Pattern Configuration Dialog: Define Tab With CustOrder Condition.....	3-63
3-26	Expression Builder: NoApproval	3-64
3-27	Expression Builder: Shipment.....	3-65
3-28	Pattern Configuration Dialog: Define Tab Complete	3-66
3-29	Measure Tab.....	3-67
3-30	Expression Builder: orderid.....	3-68
3-31	Expression Builder: amount	3-69
3-32	Measure Tab: Complete	3-70
3-33	Select Configuration Dialog: Project Tab.....	3-71
3-34	Select Configuration Dialog: Project Tab Complete	3-72
3-35	Output Configuration Dialog.....	3-73
3-36	Inject Rule Confirmation Dialog.....	3-73
3-37	CQL Rules Tab With Tracking Query.....	3-74
3-38	Stream Visualizer: Showing Missing Events	3-75
3-39	Oracle CEP Visualizer Logon Screen	3-76
3-40	Oracle CEP Visualizer Dashboard.....	3-77
3-41	CQL Application Screen: General Tab.....	3-78
3-42	CQL Application: Event Processing Network Tab	3-79
3-43	Oracle CQL Processor: General Tab.....	3-80
3-44	Oracle CQL Processor: Query Wizard Tab	3-81
3-45	Query Wizard: SSource	3-82
3-46	SSource Configuration Dialog.....	3-83
3-47	Query Wizard: Filter.....	3-84
3-48	Connecting the SSource and Filter Icons	3-84
3-49	Filter Configuration Dialog	3-85
3-50	Filter Expression Builder.....	3-86

3-51	Filter Configuration Dialog: After Adding the Filter	3-87
3-52	Query Wizard: Select.....	3-88
3-53	Select Configuration Dialog	3-89
3-54	Select Configuration Dialog: Properties Selected.....	3-90
3-55	Query Wizard: Output.....	3-91
3-56	Output Configuration Dialog.....	3-92
3-57	Inject Rule Confirmation Dialog.....	3-92
3-58	CQL Rules Tab With View StockVolGt1000	3-93
3-59	Oracle CEP Visualizer Logon Screen	3-94
3-60	Oracle CEP Visualizer Dashboard.....	3-95
3-61	CQL Application Screen: General Tab.....	3-96
3-62	CQL Application: Event Processing Network Tab	3-97
3-63	Oracle CQL Processor: General Tab.....	3-98
3-64	Oracle CQL Processor: Query Wizard Tab	3-99
3-65	Query Wizard: SSource for Moving Average Query.....	3-100
3-66	SSource Configuration Dialog: Moving Average Query	3-101
3-67	Query Wizard: Window for Moving Average Query	3-102
3-68	Window Configuration Dialog: Moving Average Query	3-103
3-69	Window Configuration Dialog: After Adding Window.....	3-104
3-70	Query Wizard: Select for Moving Average Query.....	3-105
3-71	Select Configuration Dialog: Source Property symbol Selected	3-106
3-72	Select Configuration Dialog: Source Property symbol Selected	3-107
3-73	Select Configuration Dialog: Source Property symbol Mapped to Output Event Property.... 3-108	
3-74	Select Configuration Dialog: Source Property price Selected	3-109
3-75	Expression Builder Dialog.....	3-110
3-76	Expression Builder: Applying the AVG Function.....	3-111
3-77	Select Configuration Dialog: With Expression	3-112
3-78	Select Configuration Dialog: Source Property price Mapped to Output Event Property..... 3-113	
3-79	Validation Error: GROUP BY.....	3-113
3-80	Group Tab	3-114
3-81	Group Tab: With symbol Grouping Property	3-115
3-82	Query Wizard: Output.....	3-116
3-83	Output Configuration Dialog.....	3-117
3-84	Inject Rule Confirmation Dialog.....	3-117
3-85	CQL Rules Tab With View MovingAverage	3-118
3-86	Stream Visualizer: Showing Moving Average Query Output	3-119

List of Tables

2-1	Values for the silent.xml File.....	2-9
2-2	Exit Codes	2-10
2-3	Upgrade Paths	2-11
3-1	Valid Order Workflow	3-51
3-2	Invalid Order Workflow	3-51
3-3	MATCH_RECOGNIZE Pattern Quantifiers	3-60
3-4	Condition Definitions.....	3-61

Preface

This document provides general background information and detailed code samples to help you learn about Oracle Complex Event Processing (Oracle CEP) and the Oracle Continuous Query Language (Oracle CQL).

Audience

This document is intended for users interested in learning about Oracle CEP and Oracle CQL. Readers should be familiar with basic Java development. Some knowledge of SQL would be helpful.

Documentation Accessibility

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

Accessibility of Code Examples in Documentation

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

Accessibility of Links to External Web Sites in Documentation

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

Deaf/Hard of Hearing Access to Oracle Support Services

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

Related Documents

For more information, see the following:

- *Oracle CEP Administrator's Guide*
- *Oracle CEP IDE Developer's Guide for Eclipse*
- *Oracle CEP Visualizer User's Guide*
- *Oracle CEP Java API Reference*
- *Oracle CEP CQL Language Reference*
- *Oracle CEP EPL Language Reference*
- *Oracle Database SQL Language Reference*
- SQL99 Specifications (ISO/IEC 9075-1:1999, ISO/IEC 9075-2:1999, ISO/IEC 9075-3:1999, and ISO/IEC 9075-4:1999)
- Oracle 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.

Overview of Oracle CEP

This section contains information on the following subjects:

- [Section 1.1, "Introduction to Oracle Complex Event Processing"](#)
- [Section 1.2, "Conceptual Overview of Oracle CEP"](#)
- [Section 1.3, "Event Processing Networks"](#)
- [Section 1.4, "Use Cases"](#)
- [Section 1.5, "Summary of Oracle CEP Features"](#)
- [Section 1.6, "Supported Platforms"](#)
- [Section 1.7, "Oracle CEP IDE for Eclipse"](#)
- [Section 1.8, "Oracle CEP Visualizer"](#)
- [Section 1.9, "New Features in Oracle CEP Release 11gR1 \(11.1.1\)"](#)
- [Section 1.10, "Next Steps"](#)

1.1 Introduction to Oracle Complex Event Processing

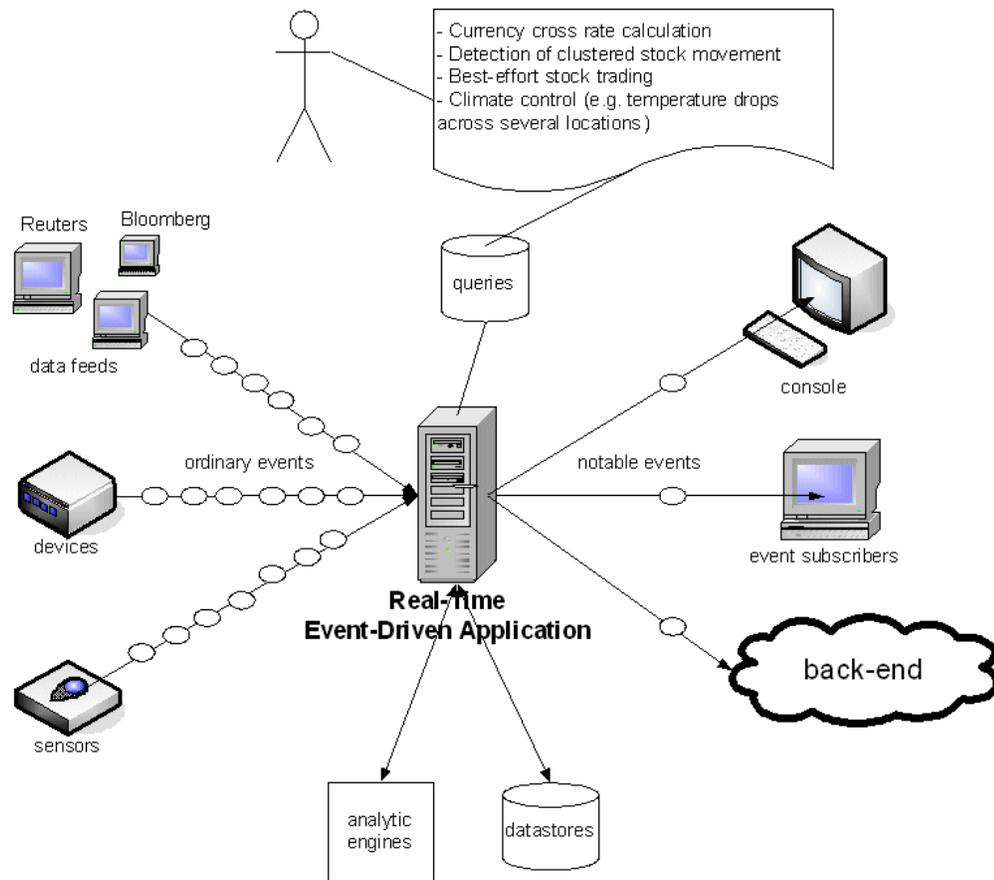
Oracle CEP (formally known as the WebLogic Event Server) is a Java server for the development and deployment of high-performance event driven applications. It is a lightweight Java application container based on Equinox OSGi, with shared services, including the Oracle CEP Service Engine, which provides 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.

Oracle CEP has the capability of deploying user Java code (POJOs) which contain the business logic. Running the business logic within Oracle CEP provides a highly tuned framework for time and event driven applications.

1.2 Conceptual Overview of Oracle CEP

[Figure 1-1](#) provides a high level view of an event-driven system.

Figure 1-1 Example Event-Driven System



An event-driven system is generally comprised of several event sources, the real-time event-driven applications, and event sinks. Oracle CEP server and the Oracle CEP applications you deploy to it comprises the event-driven applications. The event sources generate streams of ordinary event data. The Oracle CEP applications listen to the event streams, process these events, and generate notable events. Event sinks receive the notable events.

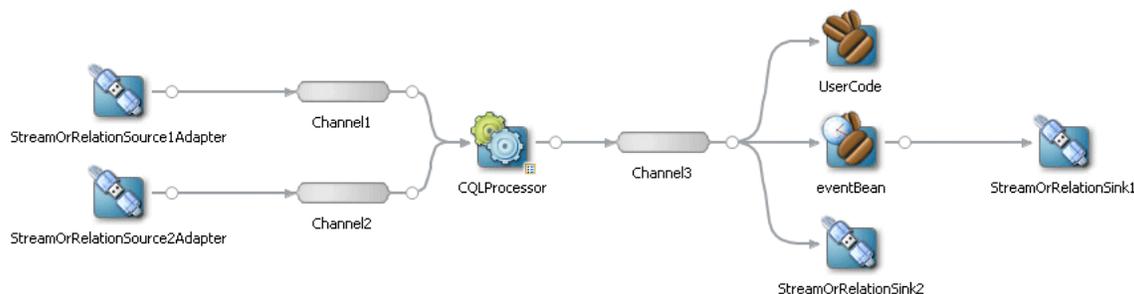
Event sources, event-driven applications, and event sinks are de-coupled from each other; one can add or remove any of these components without causing changes to the other components. This is a key attribute of event-driven architectures.

Event-driven applications are rule-driven. In Oracle CEP, rules are expressed as queries using the Oracle Continuous Query Language (Oracle CQL). These queries are persisted to a data store and are used for processing the inbound stream of events and generating the outbound stream of events. Queries typically perform filtering and aggregation functions to discover and extract notable events from the inbound event streams. As a result, the number of outbound events is generally much lower than that of the inbound events.

Oracle CEP is a middleware for the development of event-driven applications. An Oracle CEP application is essentially an event-driven application.

Next, consider the application itself, which is hosted by the Oracle CEP server, a light-weight container as shown in [Figure 1-2](#).

Figure 1–2 Oracle CEP Application



An Oracle CEP application is typically composed of the following main component types:

- **Adapters** interface directly to the inbound and outbound stream and relation sources and sinks. Adapters understand the inbound and outbound protocol, and are responsible for converting the event data into a normalized form that can be queried by a processor. Adapters forward the normalized event data into channels or outbound stream and relation sinks.
- **Channels** are event processing endpoints. Among other things, streams are responsible for queuing event data until the event processing agent can act upon it.
- **Processors** (or event processing agents) consume normalized event data from a channel, process it using queries, and may generate new events to an output channel.
- **Beans** register to listen to the output channel, and are triggered by the insertion of a new event into the output channel. This user code is generally a plain-old-Java-object (POJO). The user application makes use of a set of external services, such as JMS, Web services, and file writers, to forward the generated events to external event sinks.
- **Event Beans** register to listen to the output channel, and is triggered by the insertion of a new event into the output channel. This user code uses the Oracle CEP event bean API so that the bean can be managed by Oracle CEP.

1.3 Event Processing Networks

Adapters, channels, processors, and business logic POJOs can be connected arbitrarily to each other, forming event processing networks (EPN). Examples of topologies of EPNs are:

- Adapter - Channel - Business Logic POJO
Scenario: no processing is needed; only adaptation from proprietary protocol to some normalized model.
- Adapter - Channel - Processor - Channel - Business Logic POJO
Scenario: straight through processing to user code.
- Adapter - Channel - Processor - Channel - Business Logic POJO - Channel - Processor - Channel - Business Logic POJO
Scenario: two layers of event processing; the first processor creates causality between events and the second processor aggregates events into complex (notable) events.

EPNs have two important attributes:

- Event processing networks can be used to create a hierarchy of processing agents, and thus achieve very complex processing of events. Each layer of the EPN aggregates events of its layer into complex events that become simple events in the layer above it.
- Event processing networks improve integrability, that is, the quality of having separately developed components work correctly together. For example, one can add user code and reference to external services at several places in the network.

1.4 Use Cases

The use cases for Oracle CEP span a variety of businesses and applications. Just a few of these diverse use cases include:

- **Financial: Algorithmic Trading**
Automate stock trading based on market movement. Sample query: if, within any 20 second window, StockB rises by more than 2% and StockA does not, then automatically buy StockA.
For an example, see [Section 3.8, "Signal Generation Example"](#).
- **Transportation: Security and Fraud Detection**
Discover fraudulent activity by detecting patterns among events. Sample query: if a single ID card is used twice in less than 5 seconds to gain access to a city's subway system, alert security for piggybacking.
- **Energy and Telecommunications: Alarm Correlation**
Reduce false positive alarms. Sample query: When 15 alarms are received within any 5 second window, but less than 5 similar alarms detected within 30 seconds, then do nothing.
- **Health Care: Patient Monitoring**
Monitor the vital signs of a patient and perform some task if a particular event happens. Sample query: When a change in medication is followed by a rise in blood pressure within 20% of maximum allowable for this patient within any 10 second window, alert nearest nurse.

1.5 Summary of Oracle CEP Features

The following list summarizes the main features of Oracle CEP:

- **New in Release 11gR1 (11.1.1):** Oracle Continuous Query Language (Oracle CQL) is a query language based on SQL with added constructs that support streaming data. Using Oracle CQL, you can express queries on data streams to perform complex event processing (CEP) using Oracle CEP. Oracle CQL is scalable and comprehensive. It provides a wide range of operators (including extensive window operators), functions (including built-in, Colt, and `java.lang.Math` functions), and statements. Oracle CQL supersedes EPL.
- **New in Release 11gR1 (11.1.1):** Support for Coherence clustering and distributed cache.
- **New in Release 11gR1 (11.1.1):** Oracle CEP Visualizer query constructor, query plan generator, and support for Coherence clustering and distributed cache.

- Event Caching—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.
- Event Record and Playback—The event repository feature of Oracle CEP allows you to record events flowing through an EPN and store them so you can later play back the events.
- Built-in HTTP Publish-Subscribe Adapters—The three built-in HTTP publish-subscribe adapters allow an application to easily to publish (locally and remotely) and subscribe to an HTTP publish-subscribe server channel.
- Built-in JMS Adapters—The two JMS adapters (inbound and outbound) allow you to send and receive messages to and from a JMS queue, respectively, from your application without writing any Java code
- Oracle CEP Visualizer—A Web 2.0 application that consumes data Oracle CEP, displays it in a useful and intuitive way to system administrators and operators, and, for specified tasks, accepts data that is then passed back to Oracle CEP so as to change its configuration
- Multi-server domains (sometimes referred to as *clustering*)—Oracle CEP now allows multiple servers to be logically connected together for the purposes of management, and physically connected using a shared User Datagram Protocol (UDP) multicast address and port.
- An application server that supports deployment of Plain Old Java applications (POJOs), or Spring applications, for handling large volumes of streaming data with low latency requirements.
- Oracle CEP applications are developed and deployed as event driven applications, that is, a set of custom Spring tags is used to define the event processing network in the EPN assembly file, which extends the standard Spring context file, of your application.
- The application server contains a set of real time services that include a complex event processor (CEP), adapters, and streams. The server is highly tuned for high message throughput and low latency and deterministic behavior.
- The complex event processor is a high performance, continuous query engine for processing high volumes of streaming data. It has full support for filtering, correlation, and aggregation of streaming data from one or more streams.
- The Event Processing Language (EPL), an SQL-like language that allows event data from streams to be declaratively filtered, correlated, aggregated, and merged, with the ability to insert results into other streams for further downstream processing. You define the EPL rules either in an XML file that configures the complex event processor or programmatically using APIs. Oracle CQL supersedes EPL.
- An Adapter SDK that provides all the tools you need to create adapters that listen to incoming data feeds.
- A set of product samples that show both a simple Hello World scenario to get you started and more complex foreign exchange and algorithmic trading scenarios to showcase additional features of Oracle CEP.
- A load generator utility that simulates a data feed, useful for testing your application without needing to connect to a live data feed.
- A monitoring service that includes pre-built instrumentation for measuring throughput and latency at the component level.

- A static and dynamic configuration framework. Static configuration is performed using XML files; dynamic configuration is performed by accessing configuration and runtime MBeans using JMX and with the command-line utility `wldevs.Admin`.
- Oracle CEP is built on the Oracle microServices Architecture (mSA) which uses an OSGi-based framework to manage services provided by modules or feature sets. Oracle mSA provides the following services:
- Jetty, an HTTP container for running servlets.
- `javax.sql.DataSource` implementation and thin JDBC drivers for accessing a relational database.
- Logging and debugging.
- Authentication and authorization security.

1.6 Supported Platforms

For detailed information on the platforms that Oracle CEP supports, see:

<http://www.oracle.com/technology/software/products/ias/files/oracle%20fusion%20middleware%2011gr1%20%2811%201%201%200%29%20certification%20matrix.xls>.

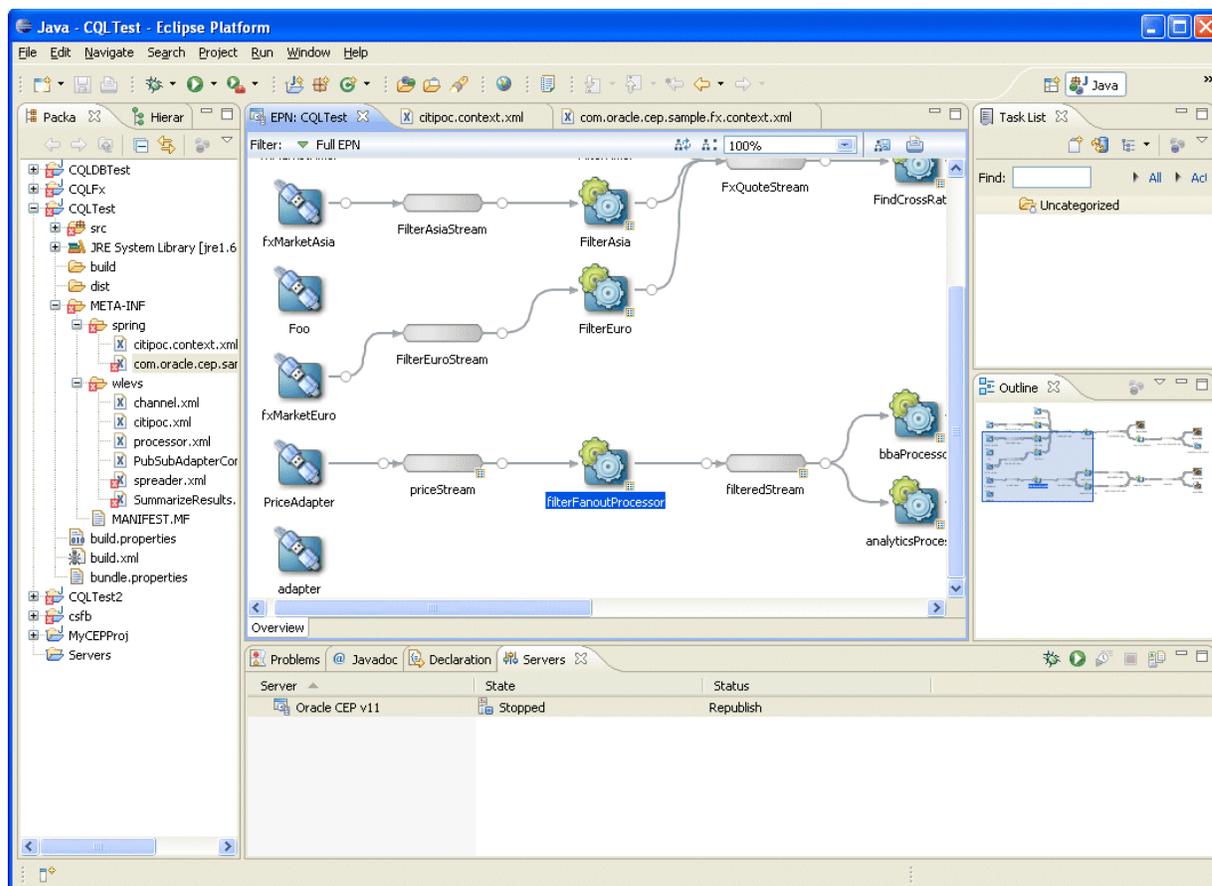
You can find the installation program appropriate for your platform here:

http://www.oracle.com/technology/software/products/middleware/html/docs/111110_fm.html. For more information, see [Section 2.2, "Installation Overview"](#).

1.7 Oracle CEP IDE for Eclipse

Oracle CEP IDE for Eclipse is targeted specifically to programmers that want to develop Oracle CEP applications as [Figure 1-3](#) shows.

Figure 1–3 Oracle CEP IDE for Eclipse



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

The key features of Oracle CEP IDE for Eclipse are:

- 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 (see [Section 1.8, "Oracle CEP Visualizer"](#)).

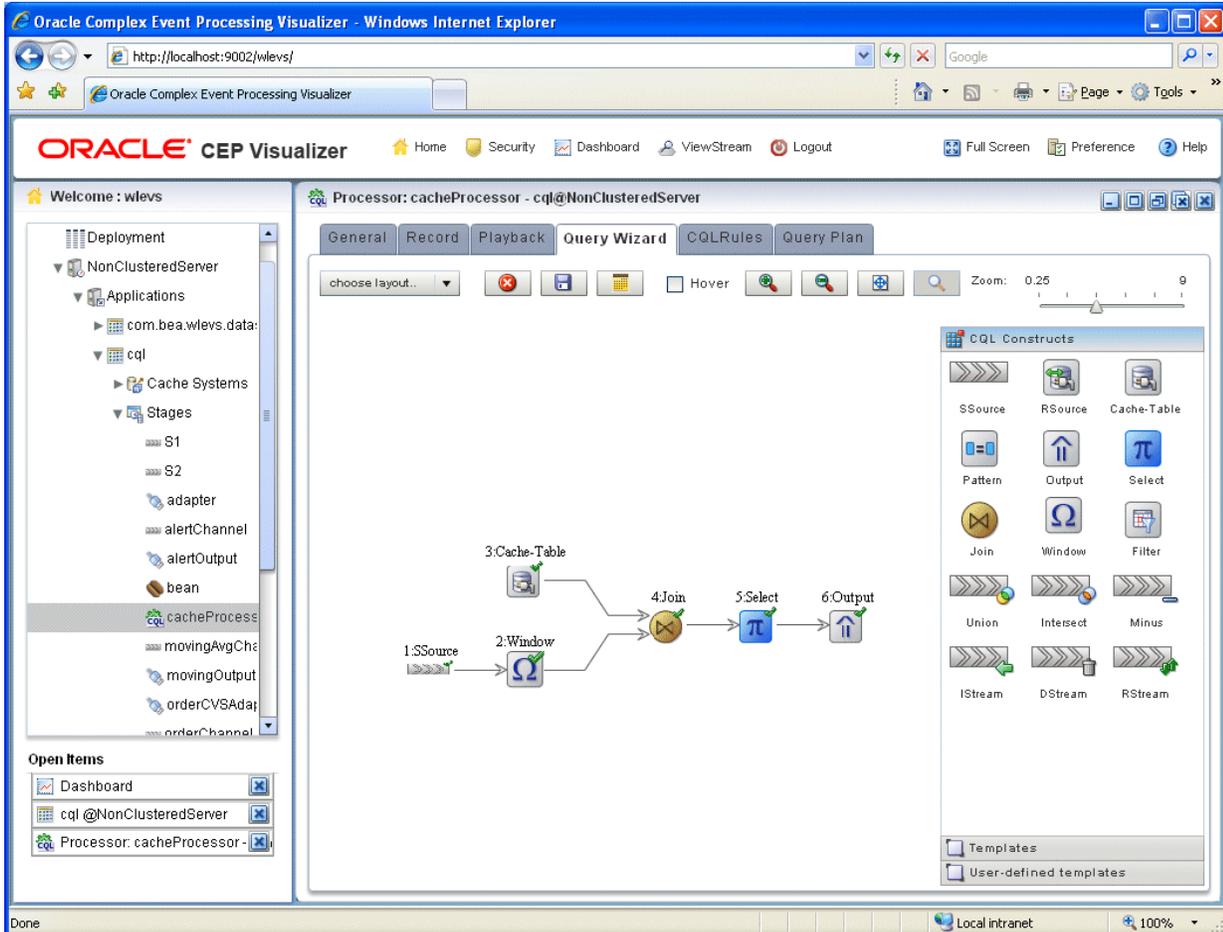
For details, see:

- *Oracle CEP IDE Developer's Guide for Eclipse*
- <http://www.oracle.com/technology/products/event-driven-architecture/cep-ide/11/index.html>

1.8 Oracle CEP Visualizer

Oracle provides an advanced run-time administration console called the Oracle CEP Visualizer as [Figure 1-4](#) shows.

Figure 1-4 Oracle CEP Visualizer



Using Oracle CEP Visualizer, you can manage, tune, and monitor Oracle CEP server domains and the Oracle CEP applications you deploy to them all from a browser. Oracle CEP Visualizer provides a variety of sophisticated run-time administration tools, including support for Oracle CQL and EPL rule maintenance and creation.

For details, see *Oracle CEP Visualizer User's Guide*

1.9 New Features in Oracle CEP Release 11gR1 (11.1.1)

This version of Oracle CEP includes the following new features:

- [Section 1.9.1, "Multi-Server Domains and Clustering"](#)
- [Section 1.9.2, "Caching"](#)
- [Section 1.9.3, "Record and Playback of Events"](#)
- [Section 1.9.4, "Visualizer Administration Console"](#)
- [Section 1.9.5, "HTTP Publish-Subscribe Adapter"](#)

- [Section 1.9.6, "JMS Adapter"](#)
- [Section 1.9.7, "Configuration Wizard for Creating Domains"](#)
- [Section 1.9.8, "Oracle Continuous Query Language \(Oracle CQL\)"](#)
- [Section 1.9.9, "Event Processing Language \(EPL\)"](#)
- [Section 1.9.10, "Security"](#)
- [Section 1.9.11, "Event Beans and Spring Beans"](#)
- [Section 1.9.12, "Suspend and Resume Applications"](#)
- [Section 1.9.13, "Domain Directory Structure Changes"](#)

1.9.1 Multi-Server Domains and Clustering

You can now configure multiple servers in an Oracle CEP domain, and cluster them together to achieve high availability, using Oracle Coherence.

See "Administrating Oracle CEP Standalone-Server Domains" in the *Oracle CEP Administrator's Guide*.

1.9.2 Caching

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 cache is a temporary storage area for events, created exclusively to improve the overall performance of your application; it is not necessary for the application to function correctly.

In this release, you can configure caching using the Oracle CEP local cache or Oracle Coherence.

See "Configuring Oracle CEP Caching" in the *Oracle CEP IDE Developer's Guide for Eclipse*.

1.9.3 Record and Playback of Events

The event repository feature of Oracle CEP allows you to record events flowing through an event processing network (EPN) and store them so you can later play back the events. The event repository is configured per stage, such as a processor or channel.

In this release, you can also schedule event record and playback.

See:

- "Recording and Playing Back Events Flowing Through an EPN" in the *Oracle CEP Visualizer User's Guide*
- "Commands for Controlling Event Record and Playback" in the *Oracle CEP Administrator's Guide*

1.9.4 Visualizer Administration Console

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

See "Overview of Oracle CEP Visualizer" in the *Oracle CEP Visualizer User's Guide*.

1.9.5 HTTP Publish-Subscribe Adapter

An HTTP Publish-Subscribe 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. Every instance of Oracle CEP includes a pub-sub server that programmers can use to implement HTTP publish-subscribe functionality in their applications.

See "Configuring HTTP Publish-Subscribe Server Adapters" in the *Oracle CEP IDE Developer's Guide for Eclipse*.

1.9.6 JMS Adapter

Oracle CEP provides both inbound and outbound JMS adapters that you can use in your event applications to send and receive messages to and from a JMS queue, respectively, without writing any Java code.

See "Configuring JMS Adapters" in the *Oracle CEP IDE Developer's Guide for Eclipse*.

1.9.7 Configuration Wizard for Creating Domains

The Configuration Wizard is a new tool for creating Oracle CEP domains.

See:

- "Creating an Oracle CEP Standalone-Server Domain" in the *Oracle CEP Administrator's Guide*
- "Creating an Oracle CEP Multi-Server Domain" in the *Oracle CEP Administrator's Guide*

1.9.8 Oracle Continuous Query Language (Oracle CQL)

Oracle Continuous Query Language (Oracle CQL) is a query language based on SQL with added constructs that support streaming data. Using Oracle CQL, you can express queries on data streams to perform complex event processing (CEP) using Oracle CEP.

Oracle CQL replaces Event Processing Language (EPL) in Oracle CEP Release 11gR1 (11.1.1). Oracle CEP supports EPL for backwards compatibility. In this release, all Oracle CEP examples are expressed in Oracle CQL.

See *Oracle CEP CQL Language Reference*.

Some Oracle CQL features include:

- Stream and relation operators—to perform the operations with streams and relations:
 - Relation-to-relation operators—to produce relations.
See "Relation-to-Relation Operators" in the *Oracle CEP CQL Language Reference*.
 - Stream-to-relation operators—to produce a relation from a stream using a variety of window operators, including:
 - * Range: time-based
S [Range T], or, optionally,
S [Range T1 Slide T2]
 - * Range: time-based unbounded

S[Range Unbounded]

- * Range: time-based now

S[Now]

- * Range: constant value

S[Range C on ID]

- * Tuple-based:

S[Rows N], or, optionally,

S[Rows N1 Slide N2]

- * Partitioned:

S[Partition By A1 ... Ak Rows N] or, optionally,

S[Partition By A1 ... Ak Rows N Range T], or

S[Partition By A1 ... Ak Rows N Range T1 Slide T2]

See "Stream-to-Relation Operators" in the *Oracle CEP CQL Language Reference*.

- Relation-to-stream operators—to produce a stream from a relation, including:

- * IStream: insert stream.

IStream(R) contains all (r, T) where r is in R at time T but r is not in R at time T-1.

- * DStream: delete stream.

DStream(R) contains all (r, T) where r is in R at time T-1 but r is not in R at time T.

- * RStream: relation stream.

RStream(R) contains all (r, T) where r is in R at time T.

See "Relation-to-Stream Operators" in the *Oracle CEP CQL Language Reference*.

- Stream-to-stream operators—to produce a stream from one or more other streams

See "Stream-to-Stream Operators" in the *Oracle CEP CQL Language Reference*.

- Queries, views, and joins:

- An Oracle CQL query is an operation that you express in Oracle CQL syntax that you execute on the Oracle CEP Service Engine to retrieve data from one or more streams or views. A top-level SELECT statement that you create using the [REGISTER|CREATE] QUERY statement is called a **query**.

In particular, you can use the Oracle CQL MATCH_RECOGNIZE condition and its various clauses to succinctly express complex conditions among stream elements to perform advanced comparisons optimized for data streams. You can use the MATCH_RECOGNIZE clause wherever Oracle CQL supports a relation variable.

Example 1–1 MATCH_RECOGNIZE Query

```
<query id="detectPerish"><![CDATA[
  select
    its.itemId
  from
    ItemTempStream
```

```

MATCH_RECOGNIZE (
  PARTITION BY
    itemId
  MEASURES
    A.itemId as itemId
  PATTERN (A B* C)
  DEFINE
    A AS (A.temp >= 25),
    B AS ((B.temp >= 25) and
          (B.element_time - A.element_time < INTERVAL "0 00:00:05:00" DAY TO
SECOND)),
    C AS (C.element_time - A.element_time >= INTERVAL "0 00:00:05:00" DAY TO
SECOND)
  ) as its
]]</query>

```

- An Oracle CQL view represents an alternative selection on a stream or relation. In Oracle CQL, you use a view instead of a subquery. A top-level VIEW statement that you create using the [REGISTER | CREATE] VIEW statement is called a **view**.
- A **join** is a query that combines rows from two or more streams, views, or relations.

See "Queries, Views, and Joins" in the *Oracle CEP CQL Language Reference*.

- Oracle CQL statements—a wide variety of statements for creating and using queries, views, functions, and windows.

See "Oracle CQL Statements" in the *Oracle CEP CQL Language Reference*.

- Functions—a wide variety of both built-in functions and base classes to extend to create user-defined functions, including:
 - single-row
 - aggregate
 - statistical and advanced arithmetic operations based on the Colt open source libraries for high performance scientific and technical computing
 - statistical and advanced arithmetic operations based on the `java.lang.Math` class

See "Functions" in the *Oracle CEP CQL Language Reference*.

- Time—control over timestamps and advanced scheduling, including:
 - application or system timestamped relations
 - application or system timestamped streams
 - derived timestamped streams

See "Time" in the *Oracle CEP CQL Language Reference*.

1.9.9 Event Processing Language (EPL)

The event processing language (EPL) has the following new features:

Note: Oracle CQL replaces Event Processing Language (EPL) in Oracle CEP Release 11gR1 (11.1.1). Oracle CEP supports EPL for backwards compatibility. For more information, see [Section 1.9.8, "Oracle Continuous Query Language \(Oracle CQL\)."](#)

- **Parameterized Queries**—Parameterized queries allow you to put placeholders inside of an EPL query in the form of a question mark. At runtime you bind these placeholders with values and they are then compiled into regular statements. The process is much like the `PreparedStatement` in JDBC.

See "Parameterized Queries" in the *Oracle CEP EPL Language Reference*.

- **Subqueries**—EPL supports both simple subqueries as well as correlated subqueries. In a simple subquery, the inner query is not correlated to the outer query. Correlated subqueries allow `SELECT` clauses to be embedded within another `SELECT` or `WHERE` clause. Both `IN` and `EXISTS` keywords are available when subqueries are used in the `WHERE` clause. Previously they were available only in the `FROM` clause and without the ability to correlate the inner and outer query.

See "Simple and Correlated Subqueries" in the *Oracle CEP EPL Language Reference*.

- **Dynamic Event Properties**—Dynamic (unchecked) properties are event properties that need not be known at statement compilation time. Such properties are resolved during runtime. The idea behind dynamic properties is that for a given underlying event representation, the properties are not always known in advance. An underlying event may have additional properties that are not known at statement compilation time, but these properties might still be required in an EPL query. The concept is especially useful for events that represent rich, object-oriented domain models or when generic container events are used.

In conjunction with dynamic event properties, the following functions have also been added to this release: `INSTANCEOF`, `CAST`, and `EXISTS`.

See "Dynamic Event Properties" in the *Oracle CEP EPL Language Reference*.

1.9.10 Security

Oracle CEP provides a variety of mechanisms to protect server resources such as data and event streams, configuration, username and password data, security policy information, remote credentials, and network traffic.

Oracle CEP supports various security providers for authentication, authorization, role mapping, and credential mapping. As initially installed, Oracle CEP is configured to use the file-based providers for both authentication and authorization. You can also configure the system to use an LDAP or DBMS provider.

In this release, you can configure custom users and groups and use them to control access to your Oracle CEP applications.

In this release, Oracle CEP supports Federal Information Processing Standards (FIPS) using the `com.rsa.jsafe.provider.JsafeJCE` security provider. Using this provider, you can configure Oracle CEP to use a FIPS-certified pseudo-random number generator for SSL.

See "Configuring Security for Oracle CEP" in the *Oracle CEP Administrator's Guide*.

1.9.11 Event Beans and Spring Beans

There is a new component type called event bean. Event beans are very similar to basic Spring beans, but they can take advantage of the full Oracle CEP framework, such as monitoring and record/playback. Event beans are functionally the same as adapters, but should be used as intermediate nodes in the EPN when adapters are used as beginning or ending nodes.

You can now also use standard Spring beans as nodes in the EPN, although the beans cannot take advantage of Oracle CEP framework.

Spring tags for event beans now include attributes for setting lifecycle callback methods via the EPN assembly file

For event beans and Spring beans that want to run in a thread, their Java class should now implement `com.bea.wlevs.ede.api.RunnableBean`.

1.9.12 Suspend and Resume Applications

Applications can now be suspended and resumed

1.9.13 Domain Directory Structure Changes

To support clustering and a multi-server environment, the directory structure of the out-of-the-box and sample domains has changed.

Previously, all domain files were located directly under the domain directory, such as `ORACLE_CEP_HOME/occep_11.1/samples/domains/helloworld_domain`, where `ORACLE_CEP_HOME` refers to the top-level installation directory of Oracle CEP.

In this release, the domain directory contains one or more sub-directories that correspond to a single server; this directory contains all the configuration files for that server. For example, the HelloWorld domain contains a single server whose configuration files are located in the following directory: `ORACLE_CEP_HOME/occep_11.1/samples/domains/helloworld_domain/defaultserver`.

The new Configuration Wizard for creating domains follows this updated directory structure.

See:

- "Creating an Oracle CEP Standalone-Server Domain" in the *Oracle CEP Administrator's Guide*
- "Creating an Oracle CEP Multi-Server Domain" in the *Oracle CEP Administrator's Guide*

1.10 Next Steps

- Install Oracle CEP Release 11gR1 (11.1.1).

See [Chapter 2, "Installing Oracle CEP."](#)

- Run the samples from their respective domains.

See:

- [Section 3.1, "Overview of the Samples Provided in the Distribution Kit"](#)
- [Section 3.6.1, "Running the HelloWorld Example from the helloworld Domain"](#)
- [Section 3.7.1, "Running the Foreign Exchange Example"](#)
- [Section 3.8.1, "Running the Signal Generation Example"](#)
- [Section 3.9.1, "Running the Event Record/Playback Example"](#)
- [Section 3.10.1, "Running the CQL Example"](#)

- Understand how the sample applications have been programmed by viewing the source and configuration files and then building them from their respective source directories.

See:

- [Section 3.6.2, "Building and Deploying the HelloWorld Example from the Source Directory"](#)
- [Section 3.7.2, "Building and Deploying the Foreign Exchange Example from the Source Directory"](#)
- [Section 3.8.2, "Building and Deploying the Signal Generation Example from the Source Directory"](#)

- Create your own Oracle CEP domain.

See:

- ["Creating an Oracle CEP Standalone-Server Domain" in the *Oracle CEP Administrator's Guide*](#)
- ["Creating an Oracle CEP Multi-Server Domain" in the *Oracle CEP Administrator's Guide*](#)
- Create a new Oracle CEP application and deploy it to your new domain.

See "Overview of Creating Oracle CEP Applications" in the *Oracle CEP IDE Developer's Guide for Eclipse*.

- Pose questions and find solutions using the Oracle CEP forum:
<http://forums.oracle.com/forums/forum.jspa?forumID=820>

Installing Oracle CEP

This section contains information on the following subjects:

- [Section 2.1, "Before You Begin"](#)
- [Section 2.2, "Installation Overview"](#)
- [Section 2.3, "Installing Oracle CEP in Graphical Mode"](#)
- [Section 2.4, "Installing Oracle CEP in Console Mode"](#)
- [Section 2.5, "Installing Oracle CEP in Silent Mode"](#)
- [Section 2.6, "Installing an Oracle CEP Patch Using Smart Update"](#)
- [Section 2.7, "Post-Installation Steps"](#)
- [Section 2.8, "Upgrading to Oracle CEP Release 11gR1 \(11.1.1\)"](#)

2.1 Before You Begin

Before you install Oracle CEP Release 11gR1 (11.1.1):

- Ensure that you are installing on a supported platform.
For more information, see [Section 1.6, "Supported Platforms"](#).
- Choose the JVM you will use:
 - If your application requires low latency, optionally install Oracle JRockit Real Time 3.1.2.

Oracle CEP performs optimally when it can access certain features from Oracle JRockit Real Time, in particular the JRockit deterministic garbage collector. Oracle CEP includes its own version of JRockit (in `ORACLE_CEP_HOME/jrockit_JAVA-VERSION_RJROCKIT-VERSION`), but it does not include the deterministic garbage collector.

Caution: Be sure you install the version of Oracle JRockit Real Time for Java version 5.0 or 6.0. Oracle JRockit Real Time for Java version 1.4.2 is not compatible with Oracle CEP Release 11gR1 (11.1.1).

For more information on Oracle JRockit Real Time, see <http://www.oracle.com/technology/products/jrockit/jrirt/index.html>.

- For other platforms (such as HP or IBM/AIX), ensure that you have the appropriate platform-specific JVM installed.

For more information, see [Section 1.6, "Supported Platforms"](#).

- Install Apache Ant, a Java-based build tool. For details, see the Apache Ant Project at <http://ant.apache.org/>.
- Decide whether or not you need to upgrade Oracle CEP and Oracle CEP applications to the current release.

For more information, see [Section 2.8, "Upgrading to Oracle CEP Release 11gR1 \(11.1.1\)"](#).

2.2 Installation Overview

You install Oracle CEP using a standard Oracle installation program.

You can find the installation program appropriate for your platform here:

http://www.oracle.com/technology/software/products/middleware/docs/111110_fm1.html. For more information on supported platforms, see [Section 1.6, "Supported Platforms"](#).

The program can be used in the following modes:

- **Graphical mode**

Graphical-mode installation is an interactive, GUI-based method for installing your software. It can be run on both Windows and UNIX systems. See [Section 2.3, "Installing Oracle CEP in Graphical Mode."](#)

Caution: If you want to run graphical-mode installation, the console attached to the machine on which you are installing the software must support a Java-based GUI. All consoles for Windows systems support Java-based GUIs, but not all consoles for UNIX systems do. If you attempt to start the installation program in graphical mode on a system that cannot support a graphical display, the installation program automatically starts console-mode installation.

- **Console mode**

Console-mode installation is an interactive, text-based method for installing your software from the command line, on either a UNIX system or a Windows system. See [Section 2.4, "Installing Oracle CEP in Console Mode."](#)

- **Silent mode**

Silent-mode installation is a non-interactive method of installing your software that requires the use of an XML properties file for selecting installation options. You can run silent-mode installation in either of two ways: as part of a script or from the command line. Silent-mode installation is a way of setting installation configurations only once and then using those configurations to duplicate the installation on many machines. See [Section 2.5, "Installing Oracle CEP in Silent Mode."](#)

2.3 Installing Oracle CEP in Graphical Mode

The Oracle CEP graphical installation program is self-explanatory, however, you can follow these steps for more information.

To install Oracle CEP in graphical mode:

1. Log in to the Windows or UNIX computer on which you want to install Oracle CEP.

Be sure you log in to the computer as the user that will be the main administrator of the Oracle CEP installation.

2. Download the product distribution file for the platform on which you want to install Oracle CEP.
3. Launch the installation program in graphical mode using the commands listed in the following table appropriate for your platform.

Platform	Instructions
Windows	Using Windows Explorer, double-click the appropriate installation program file from its download directory.
Linux or Solaris	<p>Open a command window, change to the download directory, and enter these commands:</p> <pre>prompt> chmod a+x filename prompt> ./filename</pre> <p>In the preceding commands, <i>filename</i> is the name of the installation program specific to your platform (for more information, see http://www.oracle.com/technology/software/products/middleware/htdocs/111110_fmws.html.)</p> <p>If you want to create an installation log, use the <code>-log=full_path_to_log_file</code> option; for example:</p> <pre>prompt> ./filename -log=C:\logs\server_install.log</pre>

After the installation program has finished loading, you will see the standard Welcome window.

4. Click **Next**.
5. In the Choose Home Directory window, you can specify either an existing `ORACLE_CEP_HOME` directory or create a new one.

The `ORACLE_CEP_HOME` directory is the main installation directory for Oracle CEP, such as `c:\oracle_cep`. You can have one or many `ORACLE_CEP_HOME` directories on your computer, whichever suits your development and production environments best.

If you decide to install into an existing `ORACLE_CEP_HOME` directory, the installer program checks if the directory includes the version of JRockit required by this release of Oracle CEP:

- If it finds the required JRockit installation, it does not install a new one.
- If it does not find an appropriate JRockit installation, then the program installs its own version in the `ORACLE_CEP_HOME` directory.

Use the **Browse** button to browse your computer for an existing or new `ORACLE_CEP_HOME` directory.

6. Click **Next**.
7. In the Choose Install Type window, you can choose whether to install the complete version of Oracle CEP or whether you want to pick the individual components of the product that you want to install.

Caution: By default, the complete installation does not include the product samples. If you want to install the samples (recommended), choose the **Custom** option.

8. Click **Next**.
9. If you chose Custom in the preceding step:
 - a. You will see the Choose Products and Components window. Check the components you want to install, such as the product samples.
 - b. In the JDK Selection window, select the JDKs you want to install for use with Oracle CEP. Use the **Browse** button to find other JDKs installed in different directories.
10. Click **Next**.
11. In the Choose Product Installation Directories window, you can change the default name of the home directory of Oracle CEP, `ocep_11.1`.

Although you can name this directory anything you want, Oracle recommends that you use the default name for clarity and standardization. For example, the documentation assumes that you install into the `ocep_11.1` directory.
12. Click **Next**.
13. If you are installing on Windows, and you logged in as a user with Administrator privileges, then you will see the Choose Shortcut Location window where you can choose where you want the Start Menu folder to appear. The following table describes the options available:

If you select . . . The following occurs . . .

All Users	Recommended. All users registered on the machine are provided with access to the installed software. Subsequently, if users without Administrator privileges use the Configuration Wizard from this installation to create domains, Start menu shortcuts to the domains are not created. In this case, users can manually create shortcuts in their local Start menu folders, if desired.
Local user	Other users registered on this machine will not have access to the Start menu entries for this installation.

If you logged in as a user without Administrator privileges, the Start menu entries are created in your user's local Start menu folder.

14. Click **Next**.
15. The Installation Summary window shows the products and components you are about to install, along with the approximate size in MB. This window is for your information only; to change the components to be installed, use the **Previous** button to return to the appropriate window.
16. Click **Next**.

The installer program installs Oracle CEP. The Installation Complete window indicates that the product was installed successfully.
17. Click **Done** to exit the program.
18. Review the post-installation steps that [Section 2.7, "Post-Installation Steps"](#) describes.

2.4 Installing Oracle CEP in Console Mode

Console-mode installation is an interactive, text-based method for installing your software from the command line, on either a UNIX or Windows system.

When installing in console-mode, respond to the prompts in each section by entering the number associated with your choice or by pressing Enter to accept the default. To exit the installation process, enter `exit` (or `x`, for short) in response to any prompt. To review or change your selection, enter `previous` (or `p`, for short) at the prompt. To proceed to the following window, enter `next` (or `n`, for short).

Note: In the following procedure, Windows conventions (such as back-slashes in pathnames) are used, for example, `C:\oracle_cep\ocep_11.1`. When entering pathnames on a UNIX system, be sure to use UNIX conventions, instead. For example, use forward slashes in pathnames, such as `/oracle_cep/ocep_11.1`.

To install Oracle CEP in console mode:

1. Log in to the Windows or UNIX computer on which you want to install Oracle CEP.

Be sure you log in to the computer as the user that will be the main administrator of the Oracle CEP installation.

2. Download the product distribution file for the platform on which you want to install Oracle CEP.
3. Launch the installation program in console mode using the commands listed in the following table appropriate for your platform.

Platform	Instructions
Windows	<p>Open a command window, change to the download directory, and enter the following command:</p> <pre>prompt> filename -mode=console</pre> <p>In the preceding command, <i>filename</i> is the name of the installation program specific to your platform (for more information, see http://www.oracle.com/technology/software/products/middleware/htdocs/111110_fm.html).</p> <p>If you want to create an installation log, use the <code>-log=full_path_to_log_file</code> option; for example:</p> <pre>prompt> filename -mode=console -log=C:\logs\server_install.log</pre>
Linux or Solaris	<p>Open a command window, change to the download directory, and enter these commands:</p> <pre>prompt> chmod a+x filename prompt> ./filename -mode=console</pre> <p>In the preceding commands, <i>filename</i> is the name of the installation program specific to your platform (for more information, see http://www.oracle.com/technology/software/products/middleware/htdocs/111110_fm.html).</p> <p>If you want to create an installation log, use the <code>-log=full_path_to_log_file</code> option; for example:</p> <pre>prompt> ./filename -mode=console -log=C:\logs\server_install.log</pre>

4. At the Welcome prompt, type `next` (or `n` for short) or press **Enter** to continue with the installation process.
5. In the Choose Home Directory window, the list of known home directories (if any) appear, as well as an option to create a new one.

The `ORACLE_CEP_HOME` directory is the main Oracle CEP installation directory, such as `c:\oracle_cep`. You can have one or many `ORACLE_CEP_HOME` directories on your computer, whichever suits your development and production environments best.

If you decide to install into an existing `ORACLE_CEP_HOME` directory, the installer program checks if the directory includes the version of JRockit required by this release of Oracle CEP:

- If it finds the required JRockit installation, it does not install a new one.
- If it does not find an appropriate JRockit installation, then the program installs its own version in the `ORACLE_CEP_HOME` directory.

Type `1` to create a new `ORACLE_CEP_HOME` directory, or type the number of the existing `ORACLE_CEP_HOME` directory.

6. If you chose `1` to create a new `ORACLE_CEP_HOME` directory, the installation program guides you through the required steps to create the new `ORACLE_CEP_HOME`.

Be sure to enter the full path of the `ORACLE_CEP_HOME` directory, for example `C:\oracle_cep2`.

If you specify a directory that does not exist, the installation program creates it for you.

7. In the Choose Install Type window, you can choose whether to install the complete version of Oracle CEP or whether you want to pick the individual components of the product that you want to install.

Enter `1` for a complete install or `2` for a custom install.

Caution: By default, the complete installation does not include the product samples. If you want to install the samples (recommended), choose the Custom option (`2`).

8. If you chose Custom in the preceding step, you will see the Choose Components to Install window.

Enter the numbers in brackets to toggle the components you want to install, such as the samples.

Enter `next` (or `n`) when you have chosen the components.

9. In the Choose Product Installation Directories, you can change the default name of the home directory of Oracle CEP, `ocep_11.1`, by entering a new value.

Although you can name this directory anything you want, Oracle recommends that you use the default name for clarity and standardization. For example, the documentation assumes that you install into the `ocep_11.1` directory.

Enter `next` (or `n`) when you are done.

10. If you are installing on Windows, and you logged in as a user with Administrator privileges, then you will see the Choose Shortcut Location window where you can

choose where you want the Start Menu folder to appear. The following table describes the options available:

If you select . . . The following occurs . . .	
1 "All Users"	Recommended. All users registered on the machine are provided with access to the installed software. Subsequently, if users without Administrator privileges use the Configuration Wizard from this installation to create domains, Start menu shortcuts to the domains are not created. In this case, users can manually create shortcuts in their local Start menu folders, if desired.
2 "Local user"	Other users registered on this machine will not have access to the Start menu entries for this installation.

If you logged in as a user without Administrator privileges, the Start menu entries are created in your user's local Start menu folder.

Enter the appropriate number.

The installer program installs Oracle CEP. The Installation Complete window indicates that the product was installed successfully.

11. Type `exit` to exit the program.
12. Review the post-installation steps that [Section 2.7, "Post-Installation Steps"](#) describes.

2.5 Installing Oracle CEP in Silent Mode

Silent-mode installation is a non-interactive method of installing your software that requires the use of an XML properties file for selecting installation options.

To install Oracle CEP in silent mode:

1. Log in to the Windows or UNIX computer on which you want to install Oracle CEP.
Be sure you log in to the computer as the user that will be the main administrator of the Oracle CEP installation.
2. Download the product distribution file for the platform on which you want to install Oracle CEP.
3. Create a `silent.xml` file that defines the configuration settings normally entered by a user during an interactive installation process.

See [Section 2.5.1, "Creating a silent.xml File for Silent-Mode Installation."](#)

Note: Incorrect entries in the `silent.xml` file can cause installation failures. To help you determine the cause of a failure, we recommend that you create a log file when you launch the installation program.

4. Launch the installation program in silent mode using the commands in the following table appropriate for your platform.

Platform	Instructions
Windows	<p>Open a command window, change to the download directory, and enter the following command:</p> <pre>prompt> filename -mode=silent -silent_xml=path_to_xml_file</pre> <p>In the preceding command, <i>filename</i> is the name of the installation program specific to your platform (for more information, see http://www.oracle.com/technology/software/products/middleware/htdocs/111110_fm.html) and <i>path_to_xml_file</i> is the full pathname of the <code>silent.xml</code> template file you created in the preceding step.</p> <p>If you want to create an installation log, use the <code>-log=full_path_to_log_file</code> option; for example:</p> <pre>prompt> filename -mode=silent -silent_xml=path_to_xml_file -log=C:\logs\server_install.log</pre>
Linux or Solaris	<p>Open a command window, change to the download directory, and enter these commands:</p> <pre>prompt> chmod a+x filename prompt> ./filename -mode=silent -silent_xml=path_to_xml_file</pre> <p>In the preceding commands, <i>filename</i> is the name of the installation program specific to your platform (for more information, see http://www.oracle.com/technology/software/products/middleware/htdocs/111110_fm.html) and <i>path_to_xml_file</i> is the full pathname of the <code>silent.xml</code> template file you created in the preceding step.</p> <p>If you want to create an installation log, use the <code>-log=full_path_to_log_file</code> option; for example:</p> <pre>prompt> ./filename -mode=silent -silent_xml=path_to_xml_file -log=C:\logs\server_install.log</pre>

An Oracle Installer window is displayed, indicating that the files are being extracted. No other prompt or text is displayed.

The installation is complete when the Oracle Installer window disappears.

See [Section 2.5.3, "Returning Exit Codes to the Command Window"](#) for getting information about the success or failure of the silent installation.

5. Review the post-installation steps that [Section 2.7, "Post-Installation Steps"](#) describes.

2.5.1 Creating a silent.xml File for Silent-Mode Installation

When you install Oracle CEP in silent mode, the installation program uses an XML file (`silent.xml`) to determine which installation options should be implemented.

To create a silent.xml file for silent-mode installation:

1. Using your favorite text editor, create an empty file called `silent.xml` on the computer on which you want to install Oracle CEP in silent mode.
2. Copy the contents of the sample XML file, shown in [Example 2-1](#), into your own `silent.xml` file.

Example 2-1 Sample silent.xml File for Silent-Mode Installation

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- Silent installer option: -mode=silent -silent_xml=C:\oracle\silent.xml -->
<bea-installer>
  <input-fields>
    <data-value name="BEAHOME" value="C:\oracle_cep" />
```

```

<data-value name="USER_INSTALL_DIR" value="C:\oracle_cep\ocep_11.1" />
<data-value name="INSTALL_SHORTCUT_IN_ALL_USERS_FOLDER" value="yes"/>
<data-value name="COMPONENT_PATHS" value="Oracle Complex Event Processing" />
</input-fields>
</bea-installer>

```

3. In the `silent.xml` file you just created, edit the values for the keywords shown in [Table 2–1](#) to reflect your configuration.

For example, if you want to install into the `ORACLE_CEP_HOME` directory `e:\oracle_cep`, update the corresponding `<data-value>` element as follows

```
<data-value name="BEAHOME" value="e:\oracle_cep" />
```

Table 2–1 Values for the `silent.xml` File

For this data-value name...	Enter the following value...
BEAHOME	The full pathname for the <code>ORACLE_CEP_HOME</code> directory of your choice.
USER_INSTALL_DIR	The full pathname for the directory where you want to install your Oracle CEP software.
INSTALL_SHORTCUT_IN_ALL_USERS_FOLDER	Windows only. Specify: <ul style="list-style-type: none"> ■ <code>true</code>, or <code>yes</code>, to create the shortcuts in the All Users folder. ■ <code>false</code>, or <code>no</code>, to create the shortcuts in the local users folder. <p>The user performing the installation must have Administrator privileges to install the Start menu shortcuts in the All Users folder.</p> <p>The default value for this parameter, if you do not specify it, is <code>true</code>.</p>
COMPONENT_PATHS	Specify the components and subcomponents of Oracle CEP you want to install on your system. Use the following values: <ul style="list-style-type: none"> ■ Oracle Complex Event Processing ■ Oracle Complex Event Processing/Event Server ■ Oracle Complex Event Processing/Event Server Samples <p>For additional information about entering these values, see Section 2.5.2, "Guidelines for Component Selection."</p> <p>If you do not include the <code>COMPONENT_PATHS</code> data-value name in the <code>silent.xml</code> file, the complete Oracle CEP product is installed.</p>

4. Save the file in the directory of your choice.

2.5.2 Guidelines for Component Selection

Use the following guidelines when you specify values for the `COMPONENT_PATHS` data-value name:

- When you specify a product component to be installed, all subcomponents that are installed by default in a complete installation are also installed. For example, the following entry installs both Oracle CEP and the samples:

```
<data-value name="COMPONENT_PATHS" value="Oracle Complex Event Processing" />
```

- To install multiple components or subcomponents, separate the components with a bar (`|`). Do not leave a space before or after the bar.

- To specify subcomponents, you must specify a component/subcomponent combination for each entry. For example, to explicitly install Oracle CEP and the samples, enter the following line in the file:

```
<data-value name="COMPONENT_PATHS" value="Oracle Complex Event Processing/Event Server|Oracle Complex Event Processing/Event Server Samples" />
```

Note: Because this release of Oracle CEP includes only the server itself and samples, the preceding example is equivalent to the example in the first bullet.

2.5.3 Returning Exit Codes to the Command Window

When run in silent mode, the installation program generates exit codes that indicate the success or failure of the installation. These exit codes are shown in [Table 2–2](#).

Table 2–2 Exit Codes

Code	Description
0	Installation completed successfully
-1	Installation failed due to a fatal error
-2	Installation failed due to an internal XML parsing error

[Example 2–2](#) provides a sample Windows command file that invokes the installation program in silent mode and echoes the exit codes to the command window from which the script is executed.

Example 2–2 Sample Windows Command File Displaying Silent-Mode Exit Codes

```
rem Execute the installer in silent mode
@echo off
ofm_ocep_generic_11.1.1.1.0_32_disk1_lof1.exe -mode=silent -silent_
xml=C:\downloads\silent.xml -log=C:\logs\products_silent.log

@rem Return an exit code to indicate success or failure of installation
set exit_code=%ERRORLEVEL%

@echo.
@echo Exitcode=%exit_code%
@echo.
@echo Exit Code Key
@echo -----
@echo 0=Installation completed successfully
@echo -1=Installation failed due to a fatal error
@echo -2=Installation failed due to an internal XML parsing error
@echo.
```

2.6 Installing an Oracle CEP Patch Using Smart Update

Using the Oracle Smart Update tool, you can locate, install, and activate maintenance and security updates for Oracle CEP.

For more information, see *Oracle Smart Update Installing Patches and Maintenance Packs*.

2.7 Post-Installation Steps

After installing Oracle CEP:

- Try out the product examples. For information about the examples and how to run them, see [Chapter 3, "Oracle CEP Samples."](#)
- Create your own Oracle CEP domain:

See:

- "Creating an Oracle CEP Standalone-Server Domain" in the *Oracle CEP Administrator's Guide*
- "Creating an Oracle CEP Multi-Server Domain" in the *Oracle CEP Administrator's Guide*

- Create an Oracle CEP application and deploy it to your new domain.

For a description of the programming model, details about the various components that make up an application, and how they all fit together, see "Overview of Creating Oracle CEP Applications" in the *Oracle CEP IDE Developer's Guide for Eclipse*.

2.8 Upgrading to Oracle CEP Release 11gR1 (11.1.1)

Upgrading to Oracle CEP Release 11gR1 (11.1.1) is a two-step process: first you must upgrade your applications and then you must upgrade the domain to which the applications are deployed.

[Table 2-3](#) lists the steps you must take for each supported upgrade path:

Table 2-3 Upgrade Paths

From Release	To Release 10.3	To Release 11gR1 (11.1.1)
2.0	<ol style="list-style-type: none"> 1. Section 2.8.1, "Upgrading a WebLogic Event Server 2.0 Domain to Oracle CEP 10.3" 2. Section 2.8.3, "Upgrading a WebLogic Event Server 2.0 Application to Run on Oracle CEP 10.3" 	<ol style="list-style-type: none"> 1. Section 2.8.1, "Upgrading a WebLogic Event Server 2.0 Domain to Oracle CEP 10.3" 2. Section 2.8.2, "Upgrading an Oracle CEP 10.3 Domain to Oracle CEP Release 11gR1 (11.1.1)" 3. Section 2.8.3, "Upgrading a WebLogic Event Server 2.0 Application to Run on Oracle CEP 10.3" 4. Section 2.8.4, "Upgrading an Oracle CEP 10.3 Application to Run on Oracle CEP Release 11gR1 (11.1.1)"
10.3	Not Applicable.	<ol style="list-style-type: none"> 1. Section 2.8.2, "Upgrading an Oracle CEP 10.3 Domain to Oracle CEP Release 11gR1 (11.1.1)" 2. Section 2.8.4, "Upgrading an Oracle CEP 10.3 Application to Run on Oracle CEP Release 11gR1 (11.1.1)"

For more information, see [Section 2.8.5, "Backward Compatibility Issues"](#).

2.8.1 Upgrading a WebLogic Event Server 2.0 Domain to Oracle CEP 10.3

This section describes the steps you must take to upgrade a WebLogic Event Server 2.0 domain so that it runs correctly in Oracle CEP 10.3.

For clarity, it is assumed that the existing WebLogic Event Server 2.0 domain is located in the `/bea/user_projects/domains/mydomain20` directory.

To upgrade a WebLogic Event Server 2.0 domain to Oracle CEP 10.3:

1. Using the Configuration Wizard, create a temporary Oracle CEP 10.3 domain. Later steps in this procedure require you to use or refer to files in a new Oracle CEP 10.3 domain, and it is best to use a new domain. You can later delete this domain if you want.

For the purposes of this procedure, it is assumed that the new Oracle CEP 10.3 domain is called `mydomain30`, it contains a single server called `defaultserver`, and the server files are located in the `/oracle_cep/user_projects/domains/mydomain30/defaultserver` directory.

See "Creating an Oracle CEP Standalone-Server Domain" in the *Oracle CEP Administrator's Guide*.

2. If the WebLogic Event Server 2.0 server is currently running, stop it.
3. Make a backup copy of your WebLogic Event Server 2.0 domain in case you need to revert back.
4. Replace the following two files in the WebLogic Event Server 2.0 domain with the equivalent files from the Oracle CEP 10.3 domain.
 - `lib/XACMLAuthorizerInit.ldift`
 - `lib/XACMLRoleMapperInit.ldift`

The WebLogic Event Server 2.0 files are located relative to the domain directory (`/bea/user_projects/domains/mydomain20` in our example) and the Oracle CEP 10.3 files are located relative to the server directory under the domain directory (`/oracle_cep/user_projects/domains/mydomain30/defaultserver` in our example).

5. Using your favorite text editor, open the `atnstore.txt` file in the WebLogic Event Server 2.0 domain, located in the `config` sub-directory of the main domain directory, and add the new Oracle 10.3 groups:

```
group: wlevsDeployers
description:
group: wlevsApplicationAdmins
description:
group: wlevsBusinessUsers
description:
group: wlevsOperators
description:
```

6. Remove the following files and directories (if they exist) in the WebLogic Event Server 2.0 domain:
 - `FileBasedDefaultCredentialMappermy-realmInit.initialized`
 - `FileBasedXACMLAuthorizermmy-realmInit.initialized`
 - `FileBasedXACMLRoleMappermy-realmInit.initialized`
 - `rm`
 - `cm`
 - `atz`
7. Update the `startwlevs.cmd` (Windows) or `startwlevs.sh` (Unix) command scripts in the WebLogic Event Server 2.0 domain to point to the new Oracle 10.3 binaries.

8. Update the `stopwlevs.cmd` (Windows) or `stopwlevs.sh` (Unix) command scripts in the WebLogic Event Server 2.0 domain to point to the new Oracle 10.3 binaries.
9. Start the server in the 2.0 domain using the Oracle 10.3 binaries.
"Starting and Stopping an Oracle CEP Server in a Standalone-Server Domain" in the *Oracle CEP Administrator's Guide*.
10. This upgrade procedure might have changed the security configuration of your 2.0 domain, especially if you created new users and assigned them to groups. If this is the case, use Visualizer to reconfigure the security.
See:
 - "Security Tasks" in the *Oracle CEP Visualizer User's Guide*
 - "Configuring Security for Oracle CEP" in the *Oracle CEP Administrator's Guide*

2.8.2 Upgrading an Oracle CEP 10.3 Domain to Oracle CEP Release 11gR1 (11.1.1)

This section describes the steps you must take to upgrade an Oracle CEP 10.3 domain so that it runs correctly in Oracle CEP Release 11gR1 (11.1.1).

For clarity, it is assumed that the existing Oracle CEP 10.3 domain is located in the `/bea/user_projects/domains/mydomain103` directory.

To upgrade an Oracle CEP 10.3 domain to Oracle CEP release 11gR1 (11.1.1):

1. Using the Configuration Wizard, create a temporary Oracle CEP Release 11gR1 (11.1.1) domain. Later steps in this procedure require you to use or refer to files in a new Oracle CEP Release 11gR1 (11.1.1) domain, and it is best to use a new domain. You can later delete this domain if you want.

For the purposes of this procedure, it is assumed that the new Oracle CEP Release 11gR1 (11.1.1) domain is called `mydomain11`, it contains a single server called `defaultserver`, and the server files are located in the `/oracle_cep/user_projects/domains/mydomain11/defaultserver` directory.

See "Creating an Oracle CEP Standalone-Server Domain" in the *Oracle CEP Administrator's Guide*.

2. If the WebLogic Event Server 10.3 server is currently running, stop it.
3. Make a backup copy of your WebLogic Event Server 10.3 domain in case you need to revert back.
4. Replace the following two files in the WebLogic Event Server 10.3 domain with the equivalent files from the Oracle CEP Release 11gR1 (11.1.1) domain.
 - `lib/XACMLAuthorizerInit.ldift`
 - `lib/XACMLRoleMapperInit.ldift`

The WebLogic Event Server 10.3 files are located relative to the domain directory (`/bea/user_projects/domains/mydomain30` in our example) and the Oracle CEP Release 11gR1 (11.1.1) files are located relative to the server directory under the domain directory (`/oracle_cep/user_projects/domains/mydomain11/defaultserver` in our example).

5. Using your favorite text editor, open the `atnstore.txt` file in the WebLogic Event Server 10.3 domain, located in the `config` sub-directory of the main domain directory, and add the new Oracle Release 11gR1 (11.1.1) groups:

```

group: wlevsDeployers
description:
group: wlevsApplicationAdmins
description:
group: wlevsBusinessUsers
description:
group: wlevsOperators
description:

```

6. Remove the following files and directories (if they exist) in the WebLogic Event Server 10.3 domain:
 - FileBasedDefaultCredentialMappermy-realmInit.initialized
 - FileBasedXACMLAuthorizermy-realmInit.initialized
 - FileBasedXACMLRoleMappermy-realmInit.initialized
 - rm
 - cm
 - atz
7. Update the `startwlevs.cmd` (Windows) or `startwlevs.sh` (Unix) command scripts in the WebLogic Event Server 10.3 domain to point to the new Oracle Release 11gR1 (11.1.1) binaries.
8. Update the `stopwlevs.cmd` (Windows) or `stopwlevs.sh` (Unix) command scripts in the WebLogic Event Server 10.3 domain to point to the new Oracle Release 11gR1 (11.1.1) binaries.
9. Start the server in the 10.3 domain using the Oracle Release 11gR1 (11.1.1) binaries.

"Starting and Stopping an Oracle CEP Server in a Standalone-Server Domain" in the *Oracle CEP Administrator's Guide*.
10. This upgrade procedure might have changed the security configuration of your 10.3 domain, especially if you created new users and assigned them to groups. If this is the case, use Visualizer to reconfigure the security.

See:

 - "Security Tasks" in the *Oracle CEP Visualizer User's Guide*
 - "Configuring Security for Oracle CEP" in the *Oracle CEP Administrator's Guide*

2.8.3 Upgrading a WebLogic Event Server 2.0 Application to Run on Oracle CEP 10.3

This section describes the steps you must take to upgrade an application that you developed in Version 2.0 of WebLogic Event Server so that it runs on Oracle CEP 10.3.

To upgrade a WebLogic Event Server 2.0 application to run on Oracle CEP 10.3:

1. Update the `MANIFEST.MF` file to import new versions of Spring framework and Oracle CEP packages, as well as new required packages. In particular:
 - Update the version of all imported Spring framework packages to 2.5.5. For example:

```

Import-Package:
    org.springframework.aop.framework;version="2.5.5",
    org.springframework.aop;version="2.5.5",
    ...

```

- Update the version of any imported Oracle CEP packages to 3.0.0.0. For example:

```
Import-Package:
    com.bea.wlevs.ede;version="3.0.0.0",
    com.bea.wlevs.ede.api;version="3.0.0.0",
    ...
```

- Add the following packages to the Import-Package header if they are not already included:

```
Import-Package:
    com.bea.wlevs.management.configuration.spi;version="3.0.0.0",
    com.bea.wlevs.management.spi;version="3.0.0.0",
    com.bea.wlevs.monitor;version="3.0.0.0",
    com.bea.wlevs.spi;version="3.0.0.0",
    com.bea.wlevs.spring.support;version="3.0.0.0",
    commonj.work;version="1.4.0.0",
    org.springframework.osgi.extensions.annotation;version="1.1.0",
    com.bea.wlevs.ede.spi;version="3.0.0.0",
    com.bea.wlevs.configuration.internal;version="3.0.0.0",
    ...
```

2. If you use Spring or Spring Dynamic Modules for OSGI (Spring DM) features in your application, it is possible that the declaration of the features in the Spring application context file has changed. If this is the case, you must update these declarations in the EPN assembly file of your Oracle CEP application.

Note: This change is a result of the upgrade of the Spring framework (from 2.0 to 2.5) that occurred between WebLogic Event Server 2.0 and Oracle CEP 10.3, not as a direct result of the Oracle CEP upgrade

Refer to the appropriate 2.5 XSD Schemas for any changes:

- Spring:
<http://www.springframework.org/schema/beans/spring-beans.xsd>
- Spring DM:
<http://www.springframework.org/schema/osgi/spring-osgi.xsd>

The following bullets list some of the typical changes you might have to make; the following list is not complete:

- When specifying a property to the `<osgi:service-property>` tag, use the `<entry>` tag with the key and value attributes, rather than the old `<prop>` tag.

For example, change the following 2.0 tag from:

```
<osgi:service-properties>
  <prop key="type">SocketAdapterType</prop>
</osgi:service-properties>
```

To:

```
<osgi:service-properties>
  <entry key="type" value="SocketAdapterType"/>
</osgi:service-properties>
```

- The value or ref attribute of an instance-property must always be set to an explicit value; it can no longer be an empty string to indicate an implicit use of a default value.

For example, change the following 2.0 tag from:

```
<wlevs:adapter id="fileAdapter" provider="FileAdapterType">
  <!-- file: empty value uses default
  <wlevs:instance-property name="file" value="" />
  <wlevs:listener ref="algoTradingProcessor"/>
</wlevs:adapter>
```

To:

```
<wlevs:adapter id="fileAdapter" provider="FileAdapterType">
  <wlevs:instance-property name="file" value="test.file" />
  <wlevs:listener ref="algoTradingProcessor"/>
</wlevs:adapter>
```

3. Recompile the Java code of your 2.0 adapter and business POJO implementations using your IDE. If you get compile-time errors, check the latest 10.3 Javadoc (http://download.oracle.com/docs/cd/E13157_01/wlevs/docs30/javadocs/wlevs/index.html) that describe the new Oracle CEP APIs and make the appropriate source code changes.
4. If your 2.0 application has an adapter that uses the loadgen provider as [Example 2-3](#) shows, then you must register a StockTick event type in your EPN assembly file as [Example 2-4](#) shows.

Example 2-3 Adapter Using loadgen Provider

```
<wlevs:adapter id="fxMarketAmer" provider="loadgen">
  <wlevs:instance-property name="port" value="9011"/>
</wlevs:adapter>
```

Example 2-4 Registering a StockTick Event

```
<wlevs:event-type-repository>
  <wlevs:event-type type-name="StockTick">
    <wlevs:class>com.bea.wlevs.adapter.defaultprovider.StockTickEvent</wlevs:
class>
  </wlevs:event-type>
</wlevs:event-type-repository>
```

5. After you have made the preceding changes, reassemble the application and deploy it to Oracle CEP 10.3.

See "Assembling and Deploying Oracle CEP Applications" in the *Oracle CEP IDE Developer's Guide for Eclipse*.

If, during deployment, you get an exception that indicates that a package is invisible, add this package to the Import-Package header of the MANIFEST.MF file, then reassemble and redeploy the application. Keep adding packages in this manner until the application deploys successfully.

2.8.4 Upgrading an Oracle CEP 10.3 Application to Run on Oracle CEP Release 11gR1 (11.1.1)

This section describes the steps you must take to upgrade an application that you developed in Oracle CEP 10.3 so that it runs on Oracle CEP Release 11gR1 (11.1.1).

To upgrade an Oracle CEP 10.3 application to run on Oracle CEP release 11gR1 (11.1.1):

1. Update the `MANIFEST.MF` file to import new versions of Spring framework and Oracle CEP packages, as well as new required packages.

Note that alternatively you can specify unversioned packages which will not require updating and also that you can specify larger versions in order to avoid minor version updates, that is, use "2.5" instead of "2.5.6".

In particular:

- Update the version of all imported Spring framework packages to 2.5.6. For example:

```
Import-Package:
    org.springframework.aop.framework;version="2.5.6",
    org.springframework.aop;version="2.5.6",
    ...
```

- Update the version of all imported Spring-DM framework packages to 1.2.0. For example:

```
Import-Package:
    org.springframework.osgi.context="1.2.0",
    ...
```

- Update the version of any imported Oracle CEP packages to 11.1.1.1_0. For example:

```
Import-Package:
    com.bea.wlevs.ede;version="11.1.1.1_0",
    com.bea.wlevs.ede.api;version="11.1.1.1_0",
    ...
```

- Add the following packages to the `Import-Package` header if they are not already included (see the sample source for a complete list of headers that may be required):

```
Import-Package:
    com.bea.wlevs.management.spi;version="11.1.1.1_0",
    com.bea.wlevs.spring.support;version="11.1.1.1_0",
    com.bea.wlevs.ede.spi;version="11.1.1.1_0",
    org.springframework.osgi.extensions.annotation;version="1.2.0",
    ...
```

2. If you use Spring or Spring Dynamic Modules for OSGI (Spring DM) features in your application, it is possible that the declaration of the features in the Spring application context file has changed. If this is the case, you must update these declarations in the EPN assembly file of your Oracle CEP application.

Note: This change is a result of the upgrade of the Spring-DM framework (from 1.1 to 1.2) that occurred between Oracle CEP 10.3 and Oracle CEP 11.1, not as a direct result of the Oracle CEP upgrade.

Refer to the appropriate 2.5 XSD Schemas for any changes:

- Spring:
<http://www.springframework.org/schema/beans/spring-beans.xsd>

- Spring DM:
<http://www.springframework.org/schema/osgi/spring-osgi.xsd>

In particular convert any Spring-DM declared adapter factories to use the `<wlevs:factory/>` tag instead. For example, if your 10.3 EPN assembly file contains the service that Example X shows, then you must replace this service with the `wlevs:factory` that Example Y shows.

Example 2-5 Spring-DM Declared Adapter Factory

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

Example 2-6 wlevs:factory

```
<wlevs:factory provider-name="SocketAdapterType"
 class="com.bea.wlevs.example.algotrading.adapter.SocketAdapterFactory" />
```

3. Recompile the Java code of your 10.3 adapter and business POJO implementations using your IDE. If you get compile-time errors, check the latest Release 11gR1 (11.1.1) Javadoc (see *Oracle CEP Java API Reference*) that describe the new Oracle CEP APIs and make the appropriate source code changes.

Consider changing deprecated Java API and Oracle CEP schema as the *Oracle CEP Release Notes* describe.

4. After you have made the preceding changes, reassemble the application and deploy it to Oracle CEP Release 11gR1 (11.1.1).

See "Assembling and Deploying Oracle CEP Applications" in the *Oracle CEP IDE Developer's Guide for Eclipse*.

If, during deployment, you get an exception that indicates that a package is invisible, add this package to the Import-Package header of the `MANIFEST.MF` file, then reassemble and redeploy the application. Keep adding packages in this manner until the application deploys successfully.

2.8.5 Backward Compatibility Issues

The following are non-backward compatible changes in the management framework:

- The following classes have been deprecated and removed from all operation signatures:
 - `com.bea.wlevs.management.ManagementException`
 - `com.bea.wlevs.management.ManagementRuntimeException`
 - `com.bea.wlevs.management.MbeanOperationsException`
- The following methods have been removed from all MBeans: `isRegistered()`, `preRegister()`, `postRegister()`, `getMBeanInfo()`.
- The monitoring-related methods have been removed from `StageMBean` and replaced by `com.bea.wlevs.monitor.management.MonitorRuntimeMBean`.
- The `com.bea.wlevs.management.boot.BootMBean` has been removed.

- The `com.bea.wlevs.management.configuration.ConfigSessionBean` has been removed.
- The `ObjectName` for the `AppDeploymentMBean` has been changed to include the `DomainMBean` as a parent.
- The class `com.bea.wlevs.server.management.mbean.ServerRuntimeMBean` has been changed to `com.bea.wlevs.management.runtime.ServerRuntimeMBean`.
- Two additional modules have been added: `com.bea.wlevs.management.api_*` and `com.bea.wlevs.management.spi_*`, in addition to the existing `com.bea.wlevs.management_*`.
- The service `com.bea.wlevs.spi.ManagementService` has been moved from bundle `com.bea.wlevs.spi_*` to `com.bea.wlevs.management.spi_*`.

Oracle CEP Samples

This section contains information on the following subjects:

- [Section 3.1, "Overview of the Samples Provided in the Distribution Kit"](#)
- [Section 3.2, "Installing the Samples"](#)
- [Section 3.3, "Using Oracle CEP Visualizer With the Samples"](#)
- [Section 3.4, "Increasing the Performance of the Samples"](#)
- [Section 3.5, "Setting Your Development Environment"](#)
- [Section 3.6, "HelloWorld Example"](#)
- [Section 3.7, "Foreign Exchange \(FX\) Example"](#)
- [Section 3.8, "Signal Generation Example"](#)
- [Section 3.9, "Event Record and Playback Example"](#)
- [Section 3.10, "Oracle Continuous Query Language \(CQL\) Example"](#)

3.1 Overview of the Samples Provided in the Distribution Kit

Oracle CEP includes the following samples:

- HelloWorld: a basic skeleton of a typical Oracle CEP application.
- Foreign Exchange (FX): a complete example that includes multiple components.
- Signal Generation: an example that simulates market trading and trend detection.
- Event record and playback: an example that shows how to configure event record and playback using a persistent event store.
- Oracle Continuous Query Language (CQL): an example that shows how to use the Oracle CEP Visualizer Query Wizard to construct various Oracle CQL queries to process event streams.

These samples are provided in two forms, as follows:

- [Section 3.1.1, "Ready-to-Run Samples"](#)
- [Section 3.1.2, "Sample Source"](#)

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

Note: "Additional Oracle CEP sample code can be found at http://www.oracle.com/technology/sample_code/products/event-driven-architecture".

3.1.1 Ready-to-Run Samples

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

- The sample HelloWorld domain is located in `ORACLE_CEP_HOME\ocep_11.1\samples\domains\helloworld_domain`, where `ORACLE_CEP_HOME` refers to the main Oracle CEP installation directory, such as `d:\oracle_cep`.
See [Section 3.6.1, "Running the HelloWorld Example from the helloworld Domain"](#) for details.
- The sample Foreign Exchange domain is located in `ORACLE_CEP_HOME\ocep_11.1\samples\domains\fx_domain`.
See [Section 3.7.1, "Running the Foreign Exchange Example"](#) for details.
- The sample Signal Generation domain is located in `ORACLE_CEP_HOME\ocep_11.1\samples\domains\signalgeneration_domain`.
See [Section 3.8.1, "Running the Signal Generation Example"](#) for details.
- The sample Record and Playback domain is located in `ORACLE_CEP_HOME\ocep_11.1\samples\domains\recplay_domain`.
See [Section 3.9.1, "Running the Event Record/Playback Example"](#) for details.
- The sample CQL domain is located in `ORACLE_CEP_HOME\ocep_11.1\samples\domains\cql_domain`.
See [Section 3.10.1, "Running the CQL Example"](#) for details.

3.1.2 Sample Source

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

- The HelloWorld source directory is located in `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`.
See [Section 3.6.4, "Implementation of the HelloWorld Example"](#) for details.
- The Foreign Exchange source directory is located in `ORACLE_CEP_HOME\ocep_11.1\samples\source\applications\fx`.
See [Section 3.7.4, "Implementation of the FX Example"](#) for details.
- The Signal Generation source directory is located in `ORACLE_CEP_HOME\ocep_11.1\samples\source\applications\signalgeneration`.
See [Section 3.8.4, "Implementation of the Signal Generation Example"](#) for details.
- The Record and Playback source directory is located in `ORACLE_CEP_HOME\ocep_11.1\samples\source\applications\recplay`.

See [Section 3.9.4, "Implementation of the Record and Playback Example"](#) for details.

- The CQL source directory is located in `ORACLE_CEP_HOME\ocp_11.1.1\samples\source\applications\cql`.

See [Section 3.10.4, "Implementation of the CQL Example"](#) for details.

3.2 Installing the Samples

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

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

3.3 Using Oracle CEP Visualizer With the Samples

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

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

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

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

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

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

For more information about Visualizer, see [Section 1.8, "Oracle CEP Visualizer"](#).

3.4 Increasing the Performance of the Samples

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

- Use the JRockit JDK included in Oracle JRockit Real Time 3.1.2 and enable the deterministic garbage collector by passing the `-dgc` parameter to the command that starts the Oracle CEP instance for the appropriate domain:

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

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

For more information on Oracle JRockit Real Time 3.1.2, see

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

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

3.5 Setting Your Development Environment

You must set your development environment before you can start Oracle CEP instances and run the samples. In particular, you must set the `PATH` and `JAVA_HOME` environment variables so that you are using the correct version of the JRockit JDK.

There are two ways in which JRockit might have been installed on your computer:

- As part of the Oracle JRockit Real Time 3.0 installation. This version of the JRockit JDK includes the deterministic garbage collector.
- As part of the Oracle CEP Release 11gR1 (11.1.1) installation. This version of the JRockit JDK *does not* include the deterministic garbage collector, and is provided for testing purposes only.

Although not required, Oracle recommends that you run Oracle CEP using the JRockit JDK version included in Oracle JRockit Real Time 3.0 for best results; however, the following procedures describe how to set your environment for either case.

For more information about JRockit, see [Section 3.4, "Increasing the Performance of the Samples"](#).

This section describes:

- [Section 3.5.1, "How to Set Your Development Environment on Windows"](#)
- [Section 3.5.2, "How to Set Your Development Environment on UNIX"](#)

3.5.1 How to Set Your Development Environment on Windows

This procedure describes how to set your development environment on Windows.

To make it easier to reset your development environment after logging out of a session, you can create a command file, such as `setEnv.cmd`, that contains the `set` commands this section describes.

You can also set the required environment variables permanently on your Windows computer by invoking the **Control Panel > System** window, clicking the **Advanced** tab, and then clicking the **Environment Variables** button. You can set the environment variables for the current user or for the entire system.

To set your development environment on Windows:

1. Update your `PATH` environment variable to include the `bin` directory of the JRockit JDK. Also, be sure that your `PATH` environment variable includes the `bin` directory of your Ant installation:
 - a. If using the JRockit JDK installed with Oracle JRockit Real Time 3.0:

If you installed Oracle JRockit Real Time 3.0 in the `d:\jrockit` directory and Ant is installed in the `d:\ant` directory, set your `PATH` environment variable as shown:

```
prompt> set PATH=d:\jrockit\jrvt-3.0.0-1.6.0\bin;d:\ant\bin;%PATH%
```
 - b. If using the JRockit JDK installed with Oracle CEP Release 11gR1 (11.1.1):

If you installed Oracle CEP in the `d:\oracle_cep` directory and Ant is installed in the `d:\ant` directory, set your `PATH` environment variable as shown:

```
prompt> set PATH=d:\oracle_cep\jrockit-R27.6.0-50-1.6.0_05\bin;d:\ant\bin;%PATH%
```

2. Ensure that the `JAVA_HOME` variable in the server start script points to the correct JRockit JDK. If it does not, edit the script.

The server start script (called `startwlevs.cmd`) is located in the `defaultserver` subdirectory of the main domain directory; the `defaultserver` subdirectory contains the files for the standalone server of each domain. For example, the HelloWorld domain is located in `ORACLE_CEP_HOME\ocep_11.1\samples\domains\helloworld_domain`, where `ORACLE_CEP_HOME` refers to the main Oracle CEP installation directory, such as `d:\oracle_cep`.

- a. If using the JRockit JDK installed with Oracle JRockit Real Time 3.0

The `set` command should be as follows:

```
set JAVA_HOME=d:\jrockit\jrvt-3.0.0-1.6.0
```

- b. If using the JRockit JDK installed with Oracle CEP Release 11gR1 (11.1.1):

The `set` command should be as follows:

```
set JAVA_HOME=d:\oracle_cep\jrockit-R27.6.0-50-1.6.0_05
```

3. Set the `JAVA_HOME` variable in your own development environment to point to the JRockit JDK.

- a. If using the JRockit JDK installed with Oracle JRockit Real Time 3.0:

The `set` command should be as follows:

```
prompt> set JAVA_HOME=d:\jrockit\jrvt-3.0.0-1.6.0
```

- b. If using the JRockit JDK installed with Oracle CEP Release 11gR1 (11.1.1):

The `set` command should be as follows:

```
prompt> set JAVA_HOME=d:\oracle_cep\jrockit-R27.6.0-50-1.6.0_05
```

3.5.2 How to Set Your Development Environment on UNIX

This procedure describes how to set your development environment on UNIX.

To make it easier to reset your development environment after logging out of a session, you can create a command file, such as `setEnv.sh`, that contains the `set` commands this section describes.

To set your development environment on UNIX:

1. Update your `PATH` environment variable to include the `bin` directory of the JRockit JDK. Also, be sure that your `PATH` environment variable includes the `bin` directory of your Ant installation.

- a. If using the JRockit JDK installed with Oracle JRockit Real Time 3.0:

If you installed Oracle JRockit Real Time in the `/jrockit` directory and Ant is installed in the `/ant` directory, set your `PATH` environment variable as follows:

```
prompt> PATH=/jrockit/jrvt-3.0.0-1.6.0/bin:/ant/bin:$PATH
```

- b.** If using the JRockit JDK installed with Oracle CEP 10.3

If you installed Oracle CEP in the `/oracle_cep` directory and Ant is installed in the `/ant` directory, set your `PATH` environment variable as shown:

```
prompt> PATH=/oracle_cep/jrockit-R27.6.0-50-1.6.0_05/bin:/ant/bin:$PATH
```

- 2.** Ensure that the `JAVA_HOME` variable in the server start script points to the correct JRockit JDK. If it does not, edit the script.

The server start script (called `startwlevs.sh`) is located in the `defaultserver` subdirectory of the main domain directory; the `defaultserver` subdirectory contains the files for the standalone server of each domain. For example, the HelloWorld domain is located in `ORACLE_CEP_HOME/ocp_11.1.1/samples/domains/helloworld_domain`, where `ORACLE_CEP_HOME` refers to the main Oracle CEP installation directory, such as `/oracle_cep`.

- a.** If using the JRockit JDK installed with Oracle JRockit Real Time 3.0:

The `JAVA_HOME` variable should be set as follows:

```
JAVA_HOME=/jrockit/jrvt-3.0.0-1.6.0
```

- b.** If using the JRockit JDK installed with Oracle CEP Release 11gR1 (11.1.1):

The `JAVA_HOME` variable should be set as follows:

```
JAVA_HOME=/oracle_cep/jrockit-R27.6.0-50-1.6.0_05
```

- 3.** Set the `JAVA_HOME` variable in your development environment to point to the JRockit JDK.

- a.** If using the JRockit JDK installed with Oracle JRockit Real Time 3.0:

The `JAVA_HOME` variable should be set as follows:

```
prompt> JAVA_HOME=/jrockit/jrvt-3.0.0-1.6.0
```

- b.** If using the JRockit JDK installed with Oracle CEP Release 11gR1 (11.1.1):

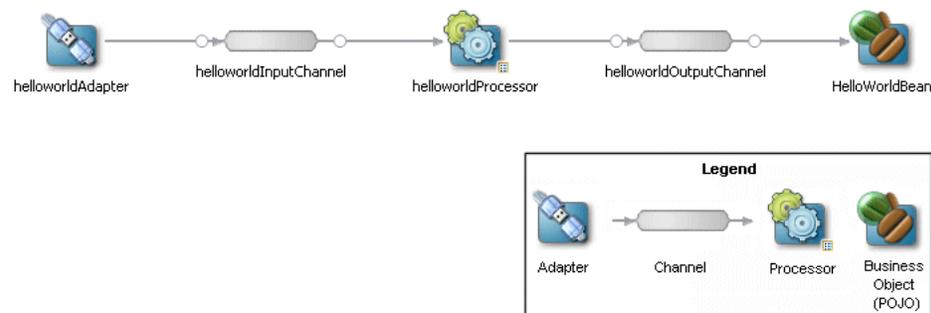
The `JAVA_HOME` variable should be set as follows:

```
prompt> JAVA_HOME=/oracle_cep/jrockit-R27.6.0-50-1.6.0_05
```

3.6 HelloWorld Example

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

[Figure 3-1](#) shows the HelloWorld example Event Processing Network (EPN). The EPN contains the components that make up the application and how they fit together.

Figure 3-1 The HelloWorld Example Event Processing Network

The example includes the following components:

- `helloworldAdapter`—Component that generates *Hello World* messages every second. In a real-world scenario, this component would typically read a stream of data from a source, such as a data feed from a financial institution, and convert it into a stream of events that the complex event processor can understand. The HelloWorld application also includes a `HelloWorldAdapterFactory` that creates instances of `HelloWorldAdapter`.
- `helloworldInputChannel`—Component that streams the events generated by the adapter (in this case *Hello World* messages) to the complex event processor.
- `helloworldProcessor`—Component that simply forwards the messages from the `helloworldAdapter` component to the POJO that contains the business logic. In a real-world scenario, this component would typically execute additional and possibly much more complex processing of the events from the stream, such as selecting a subset of events based on a property value, grouping events, and so on using Oracle CQL.
- `helloworldOutputChannel`—Component that streams the events processed by the complex event processor to the POJO that contains the user-defined business logic.
- `helloworldBean`—POJO component that simply prints out a message every time it receives a batch of messages from the processor via the output channel. In a real-world scenario, this component would contain the business logic of the application, such as running reports on the set of events from the processor, sending appropriate emails or alerts, and so on.

3.6.1 Running the HelloWorld Example from the helloworld Domain

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

To run the HelloWorld example from the helloworld domain:

1. Open a command window and change to the default server directory of the `helloworld` domain directory, located in `ORACLE_CEP_HOME\ocep_11.1\samples\domains\helloworld_domain\defaultserver`, where `ORACLE_CEP_HOME` refers to the main Oracle CEP installation directory, such as `d:\oracle_cep`.

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

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

For more information, see [Section 3.5, "Setting Your Development Environment."](#)

3. Start Oracle CEP by executing the appropriate script with the correct command line arguments:
 - a. On Windows:
 - * If you are using the JRockit JDK included in Oracle JRockit Real Time 3.0, enable the deterministic garbage collector by passing the `-dgc` parameter to the command:

```
prompt> startwlevs.cmd -dgc
```
 - * If you are not using the JRockit JDK included in Oracle JRockit Real Time 3.0:

```
prompt> startwlevs.cmd
```
 - b. On UNIX:
 - * If you are using the JRockit JDK included in Oracle JRockit Real Time 3.0, enable the deterministic garbage collector by passing the `-dgc` parameter to the command:

```
prompt> startwlevs.sh -dgc
```
 - * If you are not using the JRockit JDK included in Oracle JRockit Real Time 3.0:

```
prompt> startwlevs.sh
```

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

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

This message indicates that the HelloWorld example is running correctly.

3.6.2 Building and Deploying the HelloWorld Example from the Source Directory

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

For more information, see [Section 3.6.3, "Description of the Ant Targets to Build HelloWorld"](#).

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

1. If the helloworld Oracle CEP instance is not already running, follow the procedure in [Section 3.6.1, "Running the HelloWorld Example from the helloworld Domain"](#) to start the server.

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

2. Open a new command window and change to the HelloWorld source directory, located in `ORACLE_CEP_HOME\ocep_11.1\samples\source\applications\helloworld` where `ORACLE_CEP_HOME` is the directory in which you installed Oracle CEP.

For example:

```
prompt> cd d:\oracle_cep\ocep_11.1\samples\source\applications\helloworld
```

3. Set your development environment.

For more information, see [Section 3.5, "Setting Your Development Environment."](#)

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

```
prompt> ant all
```

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

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

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

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

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

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

3.6.3 Description of the Ant Targets to Build Hello World

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

- `clean`—This target removes the `dist` and output working directories under the current directory.
- `all`—This target cleans, compiles, and JARs up the application into a file called `com.bea.wlevs.example.helloworld_3.0.0.0.jar`, and places the generated JAR file into a `dist` directory below the current directory.
- `deploy`—This target deploys the JAR file to Oracle CEP using the Deployer utility.

For more information, see "Deployer Command-Line Reference" in the *Oracle CEP Administrator's Guide*.

3.6.4 Implementation of the HelloWorld Example

The implementation of the HelloWorld example generally follows "Creating Oracle CEP Applications: Typical Steps" in the *Oracle CEP IDE Developer's Guide for Eclipse*.

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

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

All the example files are located relative to the `ORACLE_CEP_HOME\ocp_11.1\samples\source\applications\helloworld` directory, where `ORACLE_CEP_HOME` refers to the main Oracle CEP installation directory such as `c:\oracle_cep`. Oracle recommends that you use this example directory setup in your own environment, although it is obviously not required.

The files used by the HelloWorld example include:

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

In the example, the file is called

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

For details, see [Section 3.6.5, "The HelloWorld EPN Assembly File."](#)

- Java source file for the `helloworldAdapter` component.

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

For a detailed description of this file and how to program the adapter Java files in general, see "Programming Adapters and Event Bean Classes as Event Sources: Guidelines" in the *Oracle CEP IDE Developer's Guide for Eclipse*.

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

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

For a detailed description of this file, as well as general information about programming event types, see "Creating the Event Types" in the *Oracle CEP IDE Developer's Guide for Eclipse*.

- An XML file that configures the `helloworldProcessor` and `helloworldOutputChannel` components. An important part of this file is the set of EPL rules that select the set of events that the HelloWorld application processes. You are required to include a processor configuration file in your Oracle CEP application, although the adapter and channel configuration is optional.

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

For details, see [Section 3.6.6, "The HelloWorld Component Configuration File."](#)

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

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

For a detailed description of this file, as well as general information about programming event sinks, see "Programming Event Sinks: Guidelines" in the *Oracle CEP IDE Developer's Guide for Eclipse*.

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

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

For more information about creating this file, as well as a description of creating the OSGi bundle that you deploy to Oracle CEP, see "Overview of Application Assembly and Deployment" in the *Oracle CEP IDE Developer's Guide for Eclipse*.

The HelloWorld example uses a `build.xml` Ant file to compile, assemble, and deploy the OSGi bundle; see [Section 3.6.2, "Building and Deploying the HelloWorld Example from the Source Directory"](#) for a description of this `build.xml` file if you also use Ant in your development environment.

3.6.5 The HelloWorld EPN Assembly File

One of the main purposes of the EPN assembly file is to define the event processing network by declaring the components of the application and how they are all connected, or in other words, which components listen to which other components. Oracle CEP provides a set of custom Spring tags used to declare the network. You also use the EPN assembly file to register the event types used by your application and its EPL rules.

You use the EPN assembly file in the typical way to define the application component beans in the Spring application context; the application component beans are those implemented with Java classes, such as adapters and the POJO that contains the business logic.

For more information, see:

- "Oracle CEP Application Assembly Tag Reference" in the *Oracle CEP Administrator's Guide*
- "XSD Schema Reference for Oracle CEP Files" in the *Oracle CEP Administrator's Guide*

[Example 3-1](#) shows the EPN assembly file used in the HelloWorld sample application; see the explanation after the example for details about the entries in bold.

Example 3-1 HelloWorld 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"
       xsi:schemaLocation="
http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans.xsd
http://www.springframework.org/schema/osgi
http://www.springframework.org/schema/osgi/spring-osgi.xsd
http://www.bea.com/ns/wlevs/spring
http://www.bea.com/ns/wlevs/spring/spring-wlevs-v11_0_0_0.xsd">
  <wlevs:event-type-repository>
    <wlevs:event-type type-name="HelloWorldEvent">
      <wlevs:class>com.bea.wlevs.event.example.helloworld.HelloWorldEvent</wlevs:class>
    </wlevs:event-type>
  </wlevs:event-type-repository>
  <wlevs:adapter id="helloworldAdapter"
class="com.bea.wlevs.adapter.example.helloworld.HelloWorldAdapter" >
    <wlevs:instance-property name="message" value="HelloWorld - the currenttime is:"/>
  </wlevs:adapter>
  <wlevs:processor id="helloworldProcessor" />
  <wlevs:channel id="helloworldInstream" >
    <wlevs:listener ref="helloworldProcessor"/>
    <wlevs:source ref="helloworldAdapter"/>
  </wlevs:channel>
  <wlevs:channel id="helloworldOutstream" manageable="true">
    <wlevs:listener>
      <bean class="com.bea.wlevs.example.helloworld.HelloWorldBean"/>
    </wlevs:listener>
    <wlevs:source ref="helloworldProcessor"/>
  </wlevs:channel>
</beans>
```

In the preceding example:

- The `wlevs:event-type-repository` element registers the event types that are used throughout the application; in the HelloWorld application, there is just a

single event type: `HelloWorldEvent`, implemented with the `com.bea.wlevs.event.example.helloworld>HelloWorldEvent` class. Oracle CEP automatically creates instances of this data type when needed. You can also reference this data type in the EPL rules of the application.

- The `wlevs:adapter`, `wlevs:processor`, and `wlevs:channel` elements together define the event processor network by declaring each component in the network:
 - The `wlevs:adapter` element defines the adapter component of the HelloWorld application:

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

The `id` attribute specifies a unique identifier for this component; the `id` will be referenced later by other components. The `class` attribute specifies the class that implements the adapter; in this case it is `com.bea.wlevs.adapter.example.helloworld>HelloWorldAdapter`.

The `wlevs:instance-property` child element passes an instance variable to the adapter instance; the name of the variable is `message` and the value is `HelloWorld - the current time is:`.

- The `wlevs:processor` element defines the processor component of the application:

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

The `id` attribute functions the same as that of `wlevs:adapter` element.

- The `wlevs:channel` elements define the two channel components of the application:

```
<wlevs:channel id="helloworldInstream" >
  <wlevs:listener ref="helloworldProcessor"/>
  <wlevs:source ref="helloworldAdapter"/>
</wlevs:channel>
<wlevs:channel id="helloworldOutstream" manageable="true">
  <wlevs:listener>
    <bean class="com.bea.wlevs.example.helloworld>HelloWorldBean"/>
  </wlevs:listener>
  <wlevs:source ref="helloworldProcessor"/>
</wlevs:channel>
```

The `id` attribute for streams functions the same as that of `wlevs:adapter`. The `manageable` attribute enables monitoring of the channel; by default the manageability of components is disabled.

The `wlevs:channel` element with `id="helloworldInstream"` uses the `wlevs:listener` child element to specify that the `helloworldProcessor` listens to the channel, and the `wlevs:source` child element to specify that the channel gets its events from the `helloworldAdapter` component.

The `wlevs:channel` element with `id="helloworldOutstream"` also uses these listener and source tags. One difference, however, is that it directly nests the definition of the business logic POJO in the `wlevs:listener` element rather than reference a unique identifier. In this case, the nested tag is a

standard Spring bean element that specifies that the POJO is implemented with the `com.bea.wlevs.example.helloworld.HelloWorldBean` class.

3.6.6 The HelloWorld Component Configuration File

The HelloWorld application configures the processor in the component configuration file that [Example 3–2](#) shows.

Example 3–2 HelloWorld 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>helloworldProcessor</name>
    <rules>
      <query id="helloworldRule">
        <![CDATA[ select * from helloworldInputChannel [Now] ]]>
      </query>
    </rules>
  </processor>
</n1:config>
```

If your application contains multiple processors, adapters or streams, you can either declare them all in a single configuration file, or create separate configuration files for each component; the method you chose depends on which you find easier to manage.

For each component you configure, you must add the name child element to explicitly declare the specific component to which you are referring. The value of the name element must correspond to the component's unique identifier of its declaration in the EPN assembly file.

For example, assume a processor is declared in the EPN assembly file as follows:

```
<wlevs:processor id="helloworldProcessor" ...>
```

Then its corresponding XML configuration would be as follows:

```
<processor>
  <name>helloworldProcessor</name>
  ...
</processor>
```

The HelloWorld example uses a single configuration file for one processor with the name `helloworldProcessor`. This name corresponds with the declaration of the components in the EPN assembly file.

The `processor` element configures the processor component. The most important part of the processor configuration is the declaration of the set of Oracle Continuous Query Language (Oracle CQL) rules that this processor executes; these rules select the set of events that are eventually passed to the application business object. Each rule is declared with a `query` or `relation` element using an XML `<![CDATA[...]]>` section; all `query` and `relation` elements are grouped together within a single rules element. You can define as many rules as you want for a particular processor.

The HelloWorld application has a single, very simple rule:

```
select * from helloworldInputChannel [Now]
```

This time-based range window defines its output relation such that, when $T = 0$, the relation at time t consists of tuples obtained from elements of `helloworldInputChannel` with timestamp t .

For additional information and samples about using Oracle CEP query languages, see:

- *Oracle CEP CQL Language Reference.*
- *Oracle CEP EPL Language Reference.*

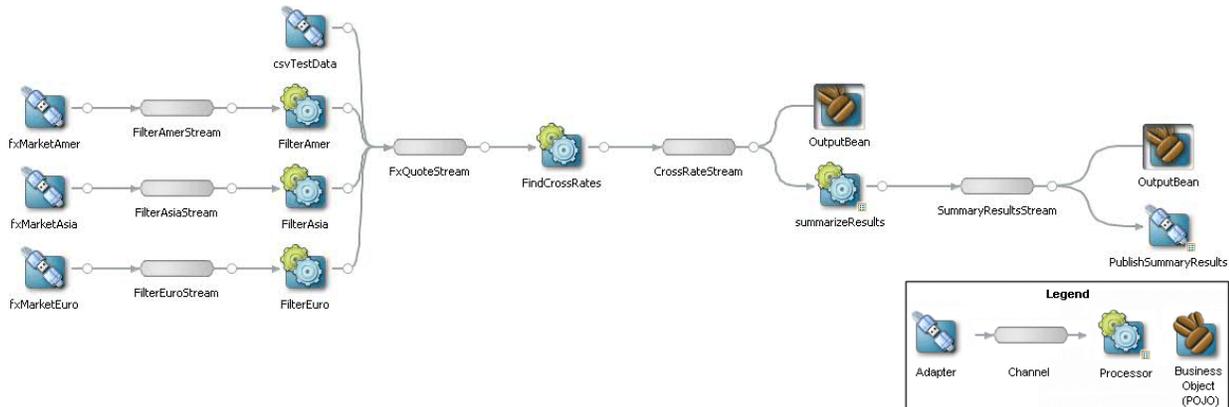
Note: Oracle EPL is superseded by Oracle CQL.

3.7 Foreign Exchange (FX) Example

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

Figure 3–2 shows the FX example Event Processing Network (EPN). The EPN contains the components that make up the application and how they fit together.

Figure 3–2 FX Example Event Processing Network



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

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

The Oracle CEP monitor is configured to watch if the event latency in the last step exceeds some threshold, such as no updated rates in a 30 second time-span, and if there is too much variance between two consecutive rates for the same currency pair.

Finally, the last rate of each currency pair is forwarded to the Oracle CEP http pub-sub server.

3.7.1 Running the Foreign Exchange Example

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

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

To run the foreign exchange example:

1. Open a command window and change to the default server directory of the FX domain directory, located in `ORACLE_CEP_HOME\ocep_11.1\samples\domains\fx_domain\defaultserver`, where `ORACLE_CEP_HOME` refers to the main Oracle CEP installation directory, such as `d:\oracle_cep`.

```
prompt> cd d:\oracle_cep\ocep_11.1\samples\domains\fx_domain\defaultserver
```

2. Set your development environment, as described in [Section 3.5, "Setting Your Development Environment."](#)
3. Start Oracle CEP by executing the appropriate script with the correct command line arguments:

a. On Windows:

- * If you are using the JRockit JDK included in Oracle JRockit Real Time 3.0, enable the deterministic garbage collector by passing the `-dgc` parameter to the command:

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

- * If you are not using the JRockit JDK included in Oracle JRockit Real Time 3.0:

```
prompt> startwlevs.cmd
```

b. On UNIX:

- * If you are using the JRockit JDK included in Oracle JRockit Real Time 3.0, enable the deterministic garbage collector by passing the `-dgc` parameter to the command:

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

- * If you are not using the JRockit JDK included in Oracle JRockit Real Time 3.0:

```
prompt> startwlevs.sh
```

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

4. To simulate an American data feed, open a new command window and set your environment as described in [Section 3.5, "Setting Your Development Environment."](#)
5. 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`.

6. Run the load generator using the `fxAmer.prop` properties file:
 - a. On Windows:

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

```
prompt> runloadgen.sh fxAmer.prop
```
7. Repeat steps 4 - 6 to simulate an Asian data feed, using the `fxAsia.prop` properties file:
 - a. On Windows:

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

```
prompt> runloadgen.sh fxAsia.prop
```
8. Repeat steps 4 - 6 to simulate an European data feed, using the `fxEuro.prop` properties file:
 - a. On Windows:

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

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

After the server status messages scroll by in the command window from which you started the server, and the three load generators start, you should see messages similar to the following being printed to the server command window:

```
OutputBean:onEvent() +
<TupleValue><EventType>SpreaderOutputEvent</EventType><ObjectName>FindCrossRates
Rule</ObjectName><Timestamp>1843704855846</Timestamp><TupleKind>null</TupleKind
><DoubleAttribute><Value>90.08350000074516</Value></DoubleAttribute><CharAttrib
ute><Value>USD</Value><Length>3</Length></CharAttribute><CharAttribute><Value>J
PY</Value><Length>3</Length></CharAttribute><IsTotalOrderGuarantee>>false</IsTot
alOrderGuarantee></TupleValue>
```

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

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

The Foreign Exchange (FX) sample source directory contains the Java source, along with other required resources such as configuration XML files, that make up the FX application. The `build.xml` Ant file contains targets to build and deploy the application to the `fx_domain` domain, as described in [Section 3.7.3, "Description of the Ant Targets to Build FX."](#)

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

1. If the FX Oracle CEP instance is not already running, follow the procedure in [Section 3.7.1, "Running the Foreign Exchange Example"](#) to start the server.

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

2. Open a new command window and change to the FX source directory, located in `ORACLE_CEP_HOME\ocep_11.1\samples\source\applications\fx`, where `ORACLE_CEP_HOME` refers to the main Oracle CEP installation directory, such as `d:\oracle_cep`.

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

3. Set your development environment, as described in [Section 3.5, "Setting Your Development Environment."](#)
4. Execute the all Ant target to compile and create the application JAR file:

```
prompt> ant all
```

5. Execute the deploy Ant target to deploy the application JAR file to Oracle CEP:

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

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

6. If the load generators required by the FX application are not running, start them as described in [Section 3.7.1, "Running the Foreign Exchange Example."](#)

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

```
{crossRate=USDJPY, internalPrice=119.09934499999781}, {crossRate=USDGBP,
internalPrice=0.503194999999915}, {crossRate=USDJPY,
internalPrice=117.73945624999783}
```

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

3.7.3 Description of the Ant Targets to Build FX

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

- `clean`—This target removes the `dist` and output working directories under the current directory.
- `all`—This target cleans, compiles, and jars up the application into a file called `com.bea.wlevs.example.fx_3.0.0.0.jar`, and places the generated JAR file into a `dist` directory below the current directory.
- `deploy`—This target deploys the JAR file to Oracle CEP using the Deployer utility.

For more information, see "Deployer Command-Line Reference" in the *Oracle CEP Administrator's Guide*.

3.7.4 Implementation of the FX Example

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

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

All the files of the FX example are located relative to the `ORACLE_CEP_HOME\ocp_11.1.1\samples\source\applications\fx` directory, where `ORACLE_CEP_HOME` refers to the main Oracle CEP installation directory such as `c:\oracle_cep`. Oracle recommends that you use this example directory setup in your own environment, although it is obviously not required.

The files used by the FX example include:

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

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

For details, see [Section 3.7.5, "The FX EPN Assembly File."](#)

- Two XML files that configure the processor components of the application:

The first XML file configures the `filterAmer`, `filterAsia`, `filterEuro`, and `FindCrossRates` processors, all in a single file. This XML file includes the Oracle CQL rules that select particular currency pairs from particular simulated market feeds and joins together all the events that were selected by the pre-processors, calculates an internal price for the particular currency pair, and then calculates the cross rate. In the example, this file is called `spreader.xml` and is located in the `META-INF/wlevs` directory.

The second XML file configures the `summarizeResults` processor and includes the Oracle CQL rule that summarizes the results of the `FindCrossRates` processor. In the example, this file is called `SummarizeResults.xml` and is located in the `META-INF/wlevs` directory.

For details, see [Section 3.7.6, "The FX Processor Component Configuration Files."](#)

- An XML file that configures the `PublishSummaryResults` `http pub-sub` adapter. In the example, this file is called `PubSubAdapterConfiguration.xml` and is located in the `META-INF/wlevs` directory.
- A Java file that implements the `OutputBean` component of the application, a POJO that contains the business logic. This POJO prints out to the screen the events that it receives, programmed in the `onEvent` method. The POJO also registers into the event type repository the `ForeignExchangeEvent` event type.

In the example, the file is called `OutputBean.java` and is located in the `src/com/oracle/cep/sample/fx` directory.

For additional information about the Oracle CEP APIs referenced in this POJO, see *Oracle CEP Java API Reference*.

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

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

For more information about creating this file, as well as a description of creating the OSGi bundle that you deploy to Oracle CEP, see "Overview of Application Assembly and Deployment" in the *Oracle CEP IDE Developer's Guide for Eclipse*.

The FX example uses a `build.xml` Ant file to compile, assemble, and deploy the OSGi bundle; see [Section 3.7.2, "Building and Deploying the Foreign Exchange Example from the Source Directory"](#) for a description of this `build.xml` file if you also use Ant in your development environment.

3.7.5 The FX EPN Assembly File

One of the main purposes of the EPN assembly file is to define the event processing network by declaring the components of the application and how they are all connected, or in other words, which components listen to which other components. Oracle CEP provides a set of custom Spring tags used to declare the network. You also use the EPN assembly file to register the event types used by your application and its Oracle CQL or EPL rules.

You use the EPN assembly file in the typical way to define the application component beans in the Spring application context; the application component beans are those implemented with Java classes, such as adapters and the POJO that contains the business logic.

For more information, see:

- "Oracle CEP Application Assembly Tag Reference" in the *Oracle CEP Administrator's Guide*
- "XSD Schema Reference for Oracle CEP Files" in the *Oracle CEP Administrator's Guide*

[Example 3-3](#) shows the EPN assembly file used in the FX sample application; see the explanation after the example for details about the entries in bold.

Example 3-3 FX 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"
       xsi:schemaLocation="
http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans.xsd
http://www.springframework.org/schema/osgi
http://www.springframework.org/schema/osgi/spring-osgi.xsd
http://www.bea.com/ns/wlevs/spring
http://www.bea.com/ns/wlevs/spring/spring-wlevs-v11_0_0_0.xsd">

  <wlevs:event-type-repository>
    <wlevs:event-type type-name="StockTick">
      <wlevs:properties>
        <wlevs:property name="lastPrice" type="double"/>
        <wlevs:property name="symbol" type="char"/>
      </wlevs:properties>
    </wlevs:event-type>
    <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 type-name="SpreaderOutputEvent">
      <wlevs:properties type="tuple">
        <wlevs:property name="internalPrice" type="double"/>
        <wlevs:property name="crossRate1" type="char"/>
        <wlevs:property name="crossRate2" type="char"/>
      </wlevs:properties>
    </wlevs:event-type>
    <wlevs:event-type type-name="CrossrateSummaryEvent">
      <wlevs:properties>
        <wlevs:property name="averageInternalPrice" type="double"/>
        <wlevs:property name="crossRatePair" type="char"/>
      </wlevs:properties>
    </wlevs:event-type>
  </wlevs:event-type-repository>

```

```
        <wlevs:property name="totalCount" type="int"/>
    </wlevs:properties>
</wlevs:event-type>
</wlevs:event-type-repository>

<!-- Assemble EPN (event processing network) -->
<wlevs:adapter id="fxMarketAmer" provider="loadgen">
    <wlevs:instance-property name="port" value="9011"/>
    <wlevs:listener ref="FilterAmerStream"/>
</wlevs:adapter>

<!-- This processor is simply here to filter the inbound event data
to make it appear more realistic. It would not normally be required
in a real application -->
<wlevs:channel id="FilterAmerStream" event-type="StockTick">
    <wlevs:listener ref="FilterAmer"/>
</wlevs:channel>
<wlevs:processor id="FilterAmer" provider="cql">
    <wlevs:listener ref="FxQuoteStream"/>
</wlevs:processor>

<wlevs:adapter id="fxMarketAsia" provider="loadgen">
    <wlevs:instance-property name="port" value="9012"/>
    <wlevs:listener ref="FilterAsiaStream"/>
</wlevs:adapter>

<!-- This processor is simply here to filter the inbound event data
to make it appear more realistic. It would not normally be required
in a real application -->
<wlevs:channel id="FilterAsiaStream" event-type="StockTick">
    <wlevs:listener ref="FilterAsia"/>
</wlevs:channel>
<wlevs:processor id="FilterAsia" provider="cql">
    <wlevs:listener ref="FxQuoteStream"/>
</wlevs:processor>

<wlevs:adapter id="fxMarketEuro" provider="loadgen">
    <wlevs:instance-property name="port" value="9013"/>
    <wlevs:listener ref="FilterEuroStream"/>
</wlevs:adapter>

<!-- This processor is simply here to filter the inbound event data
to make it appear more realistic. It would not normally be required
in a real application -->
<wlevs:channel id="FilterEuroStream" event-type="StockTick">
    <wlevs:listener ref="FilterEuro"/>
</wlevs:channel>
<wlevs:processor id="FilterEuro" provider="cql">
    <wlevs:listener ref="FxQuoteStream"/>
</wlevs:processor>

<!-- this it to allow delivery of specifc values for testing -->
<wlevs:adapter id="csvTestData" provider="csvgen">
    <wlevs:instance-property name="port" value="9014"/>
    <wlevs:instance-property name="eventName" value="StockTick"/>
    <wlevs:instance-property
        name="eventPropertyNames"
        value="symbol,lastPrice"/>
    <wlevs:listener ref="FxQuoteStream"/>
</wlevs:adapter>

<wlevs:channel id="FxQuoteStream" event-type="CrossRateEvent">
    <wlevs:listener ref="FindCrossRates"/>
</wlevs:channel>

<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>

    <wlevs:processor id="summarizeResults" provider="cql">
        <wlevs:listener ref="SummaryResultsStream"/>
    </wlevs:processor>

    <wlevs:adapter id="PublishSummaryResults" provider="httppub"></wlevs:adapter>
    <wlevs:channel id="SummaryResultsStream"
        event-type="CrossrateSummaryEvent">
        <wlevs:listener>
            <bean class="com.oracle.cep.sample.fx.OutputBean" autowire="byName"/>
        </wlevs:listener>

        <wlevs:listener ref="PublishSummaryResults"/>
    </wlevs:channel>
</beans>

```

In the preceding example:

- The `wlevs:event-type-repository` element registers the event types that are used throughout the application. These events are all of type tuple and include:
 - `StockTick`: input events.
 - `CrossRateEvent`: event type output by the `FilterAmer`, `FilterAsia`, and `FilterEuro` processors.
 - `SpreaderOutputEvent`: event type output by the `FindCrossRates` processor.
 - `CrossrateSummaryEvent`: event type output by the `summarizeResults` processor.

Oracle CEP automatically creates instances of these event types when needed. You can then reference this data in the Oracle CQL rules of the application, the adapter Java class, and the POJO.

For more information, see "Event Types" in the *Oracle CEP IDE Developer's Guide for Eclipse*.

- The set of `wlevs:adapter`, `wlevs:processor`, and `wlevs:channel` elements set up the event processor network by declaring each component in the network. The network consists of three adapters, four processors, and five streams, as described in [Figure 3-2](#).

Each component is given a unique ID which can be referenced by other components when they declare their listeners and sources.

- The `wlevs:adapter` elements specify adapter components, for example:

```

<wlevs:adapter id="fxMarketAmer" provider="loadgen">
    <wlevs:instance-property name="port" value="9011"/>
</wlevs:adapter>

```

The `fxMarketAmer`, `fxMarketAsia`, and `fxMarketEuro` adapters provide the principle input data sources. These adapters use the

`provider="loadgen"` attribute of each `wlevs:adapter` element to specify that the adapters get their data from the Oracle CEP load generator utility. The `wlevs:instance-property` child element specifies the port number to which the adapter should listen.

The `csvTestData` adapter is an optional adapter you can use to feed test data into the application. This adapter uses the `provider="csvgen"` attribute to specify that this adapter gets its data from a comma separated value (CSV) file. For more information, see "Testing Applications With the Load Generator and `csvgen` Adapter" in the *Oracle CEP IDE Developer's Guide for Eclipse*

The `PublishSummaryResults` adapter outputs events to the Oracle CEP http pub-sub server.

- The `wlevs:processor` elements specify the complex event processors, for example:

```
<wlevs:processor id="FilterAmer" provider="cql">
  <wlevs:listener ref="FxQuoteStream"/>
</wlevs:processor>
```

You can use the `listeners` attribute, common to all component elements, or a `wlevs:listener` child element to specify the components that listen to the processor; in this case, it is a channel called `FxQuoteStream`.

- The `wlevs:channel` elements specify the event streams between components, for example:

```
<wlevs:channel id="FxQuoteStream" event-type="CrossRateEvent">
  <wlevs:listener ref="FindCrossRates"/>
</wlevs:channel>
```

As with all components, you can use the `wlevs:listener` and `wlevs:source` child elements to specify the other components that act as listeners and sources for this component.

In the example, the `FindCrossRates` processor listens to the `FxQuoteStream` channel.

[Example 3–4](#) shows how you can nest the definition of a component inside a `wlevs:listener` element:

Example 3–4 Nested Component Definition

```
<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>
```

In [Example 3–4](#), the `OutputBean` POJO, declared as a standard Spring bean using the `<bean>` tag, listens to the `CrossRateStream` channel.

3.7.6 The FX Processor Component Configuration Files

The FX application uses five processors: three to handle the three data feeds, one that joins the resulting events, and one that generates summarized results.

The first four processors are configured in a single XML file, called `spreader.xml`. For more information, see [Section 3.7.6.1, "FX Processor Component Configuration File: spreader.xml."](#)

The fifth processor is configured in the XML file called `SummarizeResults.xml`. For more information, see [Section 3.7.6.2, "FX Processor Component Configuration File: SummarizeResults.xml."](#)

These XML files contain the Oracle CEP queries executed against input events. This sample uses the Oracle CQL language. For additional information and samples about using Oracle CEP query languages, see:

- *Oracle CEP CQL Language Reference.*
- *Oracle CEP EPL Language Reference.*

Note: Oracle EPL is superseded by Oracle CQL.

3.7.6.1 FX Processor Component Configuration File: spreader.xml

The first four processors are configured in a single XML file, called `spreader.xml`, as [Example 3-5](#) shows.

Example 3-5 FX Processor Component Configuration File: spreader.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<n1:config xsi:schemaLocation="http://www.bea.com/ns/wlevs/config/application
wlevs_application_config.xsd"
  xmlns:n1="http://www.bea.com/ns/wlevs/config/application"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <processor>
    <name>FilterAmer</name>
    <rules>
      <view id="UsdToEur" schema="lastPrice symbol"><![CDATA[
select lastPrice, symbol from FilterAmerStream [range 1] where
symbol="USDEUR"
]]></view>
      <view id="UsdToEurPre" schema="price fromRate toRate"><![CDATA[
select avg(lastPrice) as price, "USD" as fromRate, "EUR" as toRate
from UsdToEur
where lastPrice < 3.0 and lastPrice > 0.25
]]></view>
      <query id="AmerFilterCrossRates"><![CDATA[select * from
UsdToEurPre]]></query>
    </rules>
  </processor>
  <processor>
    <name>FilterAsia</name>
    <rules>
      <view id="EurToJpy" schema="lastPrice symbol"><![CDATA[
select lastPrice, symbol from FilterAsiaStream [range 1] where
symbol="EURJPY"
]]></view>
      <view id="EurToJpyPre" schema="price fromRate toRate"><![CDATA[
select avg(lastPrice) as price, "EUR" as fromRate, "JPY" as toRate
from EurToJpy
where lastPrice < 200.0 and lastPrice > 100.0
]]></view>
      <query id="AsiaFilterCrossRates"><![CDATA[select * from
```

```

EurToJpyPre]]></query>
    </rules>
</processor>
<processor>
    <name>FilterEuro</name>
    <rules>
        <view id="EurToGbp" schema="lastPrice symbol"><![CDATA[
            select lastPrice, symbol from FilterEuroStream [range 1] where
symbol="EURGBP"
        ]]></view>
        <view id="EurToGbpPre" schema="price fromRate toRate"><![CDATA[
            select avg(lastPrice) as price, "EUR" as fromRate, "GBP" as toRate
            from EurToGbp
            where lastPrice < 1.5 and lastPrice > 0.5
        ]]></view>
        <query id="EuroFilterCrossRates"><![CDATA[select * from
EurToGbpPre]]></query>
    </rules>
</processor>

<processor>
    <name>FindCrossRates</name>
    <rules>
        <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>
    </rules>
</processor>
</nl:config>

```

The `FilterAmer`, `FilterAsia`, and `FilterEuro` processors in this file are all essentially the same; the differences lie only in the values used in the Oracle CQL queries for querying different items from the data feeds and applying different boundary conditions. For this reason, this section will discuss just the `FilterAmer` processor.

The Oracle CQL rules executed by the `FilterAmer` processor are:

```

<processor>
    <name>FilterAmer</name>
    <rules>
        <view id="UsdToEur" schema="lastPrice symbol"><![CDATA[
            select lastPrice, symbol from FilterAmerStream [range 1] where
symbol="USDEUR"
        ]]></view>
        <view id="UsdToEurPre" schema="price fromRate toRate"><![CDATA[
            select avg(lastPrice) as price, "USD" as fromRate, "EUR" as toRate
            from UsdToEur
            where lastPrice < 3.0 and lastPrice > 0.25
        ]]></view>
        <query id="AmerFilterCrossRates"><![CDATA[select * from
UsdToEurPre]]></query>

```

```

    </rules>
  </processor>

```

To understand the query, one must look at the various clauses, as follows:

- The `UsdToEur` view serves as a subquery that selects only those items from the `StockTick` data feed in which the `symbol` value is `USDEUR` (US dollar - European euro exchange) and should reject all other items. The `from` clause specifies also specifies that the window of time for which this Oracle CQL query executes is 1 second.
- The `UsdToEurPre` view serves as a subquery that selects from the `UsdToEur` view. Its `where` clause specifies the boundary condition to ensure that the rates for a particular item from the feed fall within an accepted range; in this case, the `lastPrice` for a particular item from the feed must be between \$3.00 and \$0.25.
- The `AmerFilterCrossRates` query selects everything from the `UsdToEurPre` view. Only the results of a query are output to the down-stream channels that listen to a processor.

The Oracle CQL rule executed by the `FindCrossRates` processor are:

```

<processor>
  <name>FindCrossRates</name>
  <rules>
    <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>
  </rules>
</processor>

```

To understand the query, one must look at the various clauses, as follows:

- The `from` and `where` clauses join two events from the `CrossRateEvent` object (which contains events selected by the `FilterAmer`, `FilterAsia`, and `FilterEuro` processors) where the value of the `toRate` and `fromRate` are the same. The `from` clause also sets the processing window, again of 1 second.
- The `select` clause calculates an internal price of a particular currency, which averages the `to` and `from` rate of a the currency plus a fee of \$.05, and also calculates a cross rate, which is defined as the price of one currency in terms of another currency in the market of a third country.

The result of this query is then sent to the business object `POJO` and the `summarizeResults` processor.

3.7.6.2 FX Processor Component Configuration File: `SummarizeResults.xml`

The fifth processor is configured in the XML file called `SummarizeResults.xml` as [Example 3–6](#) shows.

Example 3–6 FX Processor Component Configuration File: SummarizeResults.xml

```

<?xml version="1.0" encoding="UTF-8"?>
<wlevs:config xmlns:wlevs="http://www.bea.com/ns/wlevs/config/application">
  <processor>
    <name>summarizeResults</name>
    <rules>
      <query id="Rule"><![CDATA[
        select
          crossRate1 || crossRate2 as crossRatePair,
          count(*) as totalCount,
          avg(internalPrice) as averageInternalPrice
        from CrossRateStream [RANGE 1 SLIDE 1]
        group by crossRate1,crossRate2
        having count(*) > 0
      ]]></query>
    </rules>
  </processor>
</wlevs:config>

```

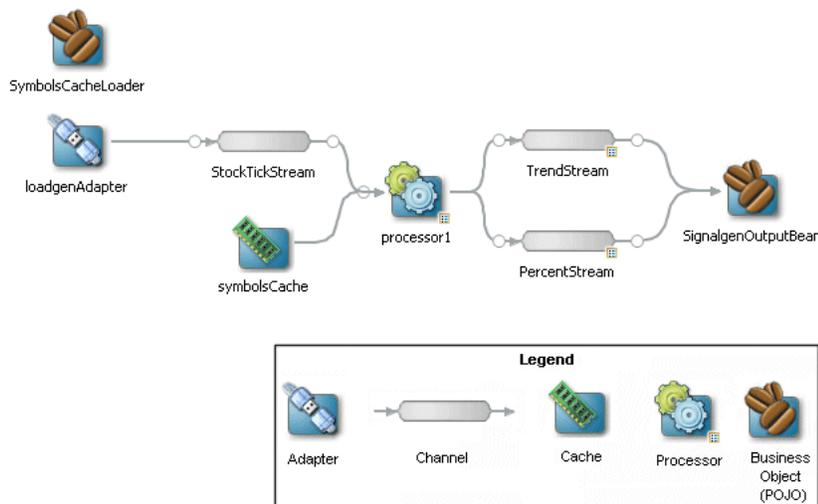
The result of this query is then sent to the business object POJO and the `PublishSummaryResults` adapter.

3.8 Signal Generation Example

The signal generation sample application receives simulated market data and verifies if the price of a security has fluctuated more than two percent. The application also detects if there is a *trend* occurring by keeping track of successive stock prices for a particular symbol; if more than three successive prices fluctuate more than two percent, this is considered a trend.

Figure 3–3 shows the signal generation example Event Processing Network (EPN). The EPN contains the components that make up the application and how they fit together.

Figure 3–3 The Signal Generation Example Event Processing Network



The application simulates a market data feed using the Oracle CEP load generator utility; in this example, the load generator generates up to 10,000 messages per second. The example includes an HTML dashboard which displays the matched events along with the latencies; events consist of a stock symbol, a timestamp, and the price.

The example demonstrates very low latencies, with minimum latency *jitter* under high throughputs. Once the application starts running, the processor matches an average of 800 messages per second. If the application is run on the minimum configured system, the example shows very low average latencies (30-300 microsecond, on average) with minimal latency spikes (low milliseconds).

The example computes and displays latency values based on the difference between a timestamp generated on the load generator and timestamp on Oracle CEP. Computing valid latencies requires very tight clock synchronization, such as 1 millisecond, between the computer running the load generator and the computer running Oracle CEP. For this reason, Oracle recommends running both the load generator and Oracle CEP on a single multi-CPU computer where they will share a common clock.

The example also shows how to use the Oracle CEP event caching feature. In particular the single processor in the EPN sends events to both an event bean and a cache.

The example also demonstrates how to use Oracle CQL queries.

3.8.1 Running the Signal Generation Example

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

The `signalgeneration_domain` domain contains a single application: the signal generation sample application. To run the signal generation application, you simply start an instance of Oracle CEP in that domain.

To run the signal generation example:

1. Open a command window and change to the default server directory of the `signalgeneration_domain` domain directory, located in `ORACLE_CEP_HOME\ocep_11.1\samples\domains\signalgeneration_domain\defaultserver`, where `ORACLE_CEP_HOME` refers to the main Oracle CEP installation directory, such as `d:\oracle_cep`.

```
prompt> cd d:\oracle_cep\ocep_11.1\samples\domains\signalgeneration_
domain\defaultserver
```

2. Set your development environment, as described in [Section 3.5, "Setting Your Development Environment."](#)
3. Start Oracle CEP by executing the appropriate script with the correct command line arguments:
 - a. On Windows:
 - * If you are using the JRockit JDK included in Oracle JRockit Real Time 3.0, enable the deterministic garbage collector by passing the `-dgc` parameter to the command:


```
prompt> startwlevs.cmd -dgc
```
 - * If you are not using the JRockit JDK included in Oracle JRockit Real Time 3.0:


```
prompt> startwlevs.cmd
```
 - b. On UNIX:

- * If you are using the JRockit JDK included in Oracle JRockit Real Time 3.0, enable the deterministic garbage collector by passing the `-dgc` parameter to the command:

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

- * If you are not using the JRockit JDK included in Oracle JRockit Real Time 3.0:

```
prompt> startwlevs.sh
```

4. Wait until you see console messages like this:

```
<Apr 24, 2009 11:40:37 AM EDT> <Notice> <Server> <BEA-2046000> <Server STARTED>
Throughput (msg per second): 0. Average latency (microseconds): 0
Throughput (msg per second): 0. Average latency (microseconds): 0
Throughput (msg per second): 0. Average latency (microseconds): 0
Throughput (msg per second): 0. Average latency (microseconds): 0
...
```

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

Next, to simulate a data feed, you use a load generator programmed specifically for the example.

5. Open a new command window.
6. Change to the `ORACLE_CEP_HOME\ocep_11.1\samples\domains\signalgeneration_domain\defaultserver\utils` directory, where `ORACLE_CEP_HOME` refers to the main Oracle CEP installation directory, such as `d:\oracle_cep`.
7. Run the `startDataFeed` command:

- a. On Windows:

```
prompt> startDataFeed.cmd
```

- b. On UNIX:

```
prompt> startDataFeed.sh
```

8. Invoke the example dashboard by starting a browser and opening the following HTML page:

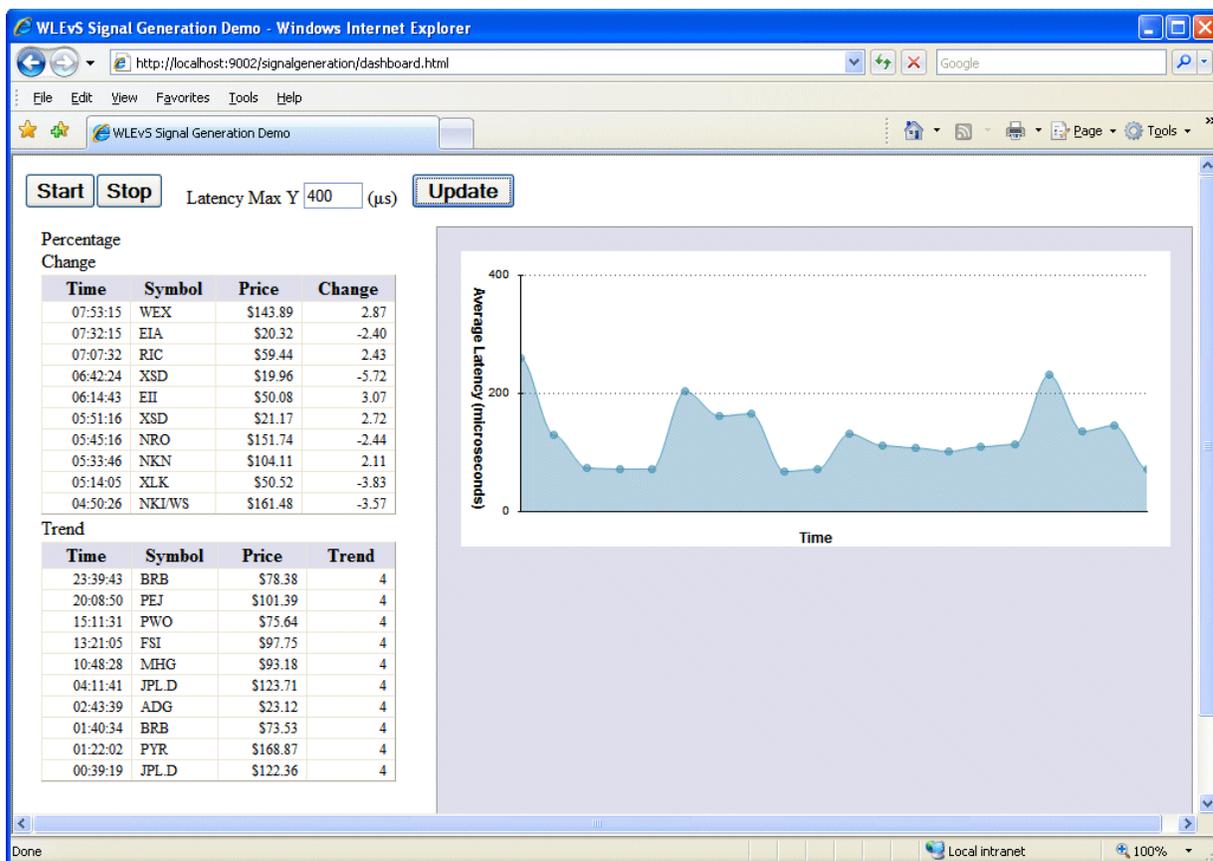
```
http://host:9002/signalgeneration/dashboard.html
```

Replace `host` with the name of the computer on which Oracle CEP is running; if it is the same computer as your browser, you can use `localhost`.

9. In the browser, click **Start** on the HTML page.

You should start seeing the events that match the Oracle CQL rules configured for this example as [Figure 3-4](#) shows.

Figure 3–4 Signal Generation Dashboard



3.8.2 Building and Deploying the Signal Generation Example from the Source Directory

The signal generation sample source directory contains the Java source, along with other required resources, such as configuration XML files, EPN assembly file, and DOJO client Javascript libraries, that make up the signal generation application. The build.xml Ant file contains targets to build and deploy the application to the signalgeneration_domain domain, as described in [Section 3.8.3, "Description of the Ant Targets to Build Signal Generation."](#)

To build and deploy the signal generation example from the source directory:

1. If the signal generation Oracle CEP instance is not already running, follow the procedure in [Section 3.8.1, "Running the Signal Generation Example"](#) to start the server. You must have a running server to successfully deploy the rebuilt application.
2. Open a new command window and change to the signal generation source directory, located in `ORACLE_CEP_HOME\ocep_11.1\samples\source\applications\signalgeneration`, where `ORACLE_CEP_HOME` refers to the main Oracle CEP installation directory, such as `d:\oracle_cep`.

```
prompt> cd d:\oracle_cep\ocep_11.1\samples\source\applications\signalgeneration
```
3. Set your development environment, as described in [Section 3.5, "Setting Your Development Environment."](#)
4. Execute the all Ant target to compile and create the application JAR file:

```
prompt> ant all
```

5. Execute the `deploy` Ant target to deploy the application JAR file to the `ORACLE_CEP_HOME\ocep_11.1\samples\domains\signalgeneration_domain\defaultserver\applications\signalgeneration` directory:

```
prompt> ant deploy
```

Caution: This target overwrites the existing signal generation application JAR file in the domain directory.

6. If the load generator required by the signal generation application is not running, start it as described in [Section 3.8.1, "Running the Signal Generation Example."](#)
7. Invoke the example dashboard as described in [Section 3.8.1, "Running the Signal Generation Example."](#)

3.8.3 Description of the Ant Targets to Build Signal Generation

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

- `clean`—This target removes the `dist` and `output` working directories under the current directory.
- `all`—This target cleans, compiles, and jars up the application into a file called `com.bea.wlevs.example.signalgen_3.0.0.0.jar`, and places the generated JAR file into a `dist` directory below the current directory.
- `deploy`—This target deploys the JAR file to Oracle CEP using the Deployer utility.

For more information, see "Deployer Command-Line Reference" in the *Oracle CEP Administrator's Guide*.

3.8.4 Implementation of the Signal Generation Example

The implementation of the signal generation example generally follows "Creating Oracle CEP Applications: Typical Steps" in the *Oracle CEP IDE Developer's Guide for Eclipse*.

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

All the files of the signal generation are located relative to the `ORACLE_CEP_HOME\ocep_11.1\samples\source\applications\signalgeneration` directory, where `ORACLE_CEP_HOME` refers to the main Oracle CEP installation directory such as `c:\oracle_cep`. Oracle recommends that you use this example directory setup in your own environment, although it is obviously not required.

The files used by the signal generation example include:

- A EPN assembly file that describes each component in the application and how all the components are connected together.

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

For details, see [Section 3.8.5, "The Signal Generation EPN Assembly File."](#)

- An XML file that configures the processor component of the application; this file is called `config.xml` and is located in the `META-INF/wlevs` directory

The `config.xml` file configures the `processor1` Oracle CQL processor, in particular the Oracle CQL rules that verify whether the price of a security has fluctuated more than two percent and whether a trend has occurred in its price.

For details, see [Section 3.8.6, "The Signal Generation Component Configuration Files."](#)

- A Java file that implements the `SignalgenOutputBean` component of the application, a POJO that contains the business logic. This POJO is an `HttpServlet` and an `EventSink`. Its `onEvent` method consumes `PercentTick` and `TrendTick` event instances, computes latency, and displays dashboard information.

In the example, the file is called `SignalgenOutputBean.java` and is located in the `src/oracle/cep/example/signalgen` directory.

For a detailed description of this file, as well as general information about programming event sinks, see "Stream Sources and Stream Sinks and Relation Sources and Relation Sinks" in the *Oracle CEP IDE Developer's Guide for Eclipse*.

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

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

For more information about creating this file, as well as a description of creating the OSGi bundle that you deploy to Oracle CEP, see "Overview of Application Assembly and Deployment" in the *Oracle CEP IDE Developer's Guide for Eclipse*.

- A `dashboard.html` file in the main example directory; this HTML file is the example dashboard that displays events and latencies of the running signal generation application. The HTML file uses Dojo Javascript libraries from <http://dojotoolkit.org/>, located in the `dojo` directory.

For additional information about the Oracle CEP APIs referenced in `ForeignExchangeBuilderFactory`, see *Oracle CEP Java API Reference*.

The signal generation example uses a `build.xml` Ant file to compile, assemble, and deploy the OSGi bundle; see [Section 3.8.2, "Building and Deploying the Signal Generation Example from the Source Directory"](#) for a description of this `build.xml` file if you also use Ant in your development environment.

3.8.5 The Signal Generation EPN Assembly File

One of the main purposes of the EPN assembly file is to define the event processing network by declaring the components of the application and how they are all connected, or in other words, which components listen to which other components. Oracle CEP provides a set of custom Spring tags used to declare the network. You also use the EPN assembly file to register the event types used by your application and its Oracle CQL rules.

You use the EPN assembly file in the typical way to define the application component beans in the Spring application context; the application component beans are those implemented with Java classes, such as adapters and the POJO that contains the business logic.

For more information, see:

- "Schema Reference: EPN Assembly spring-wlevs-v11_0_0_0.xsd" in the *Oracle CEP IDE Developer's Guide for Eclipse*
- "Oracle CEP Schemas" in the *Oracle CEP IDE Developer's Guide for Eclipse*

Example 3-7 shows the EPN assembly file used in the signal generation sample application; see the explanation after the example for details about the entries in bold.

Example 3-7 Signal Generation 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_0_0_
0.xsd">

  <wlevs:event-type-repository>
    <wlevs:event-type type-name="TrendTick">
      <wlevs:properties>
        <wlevs:property name="symbol" type="char"/>
        <wlevs:property name="lastPrice" type="double"/>
        <wlevs:property name="trendLastPrice" type="bigint"/>
        <wlevs:property name="startTimestamp" type="bigint"/>
      </wlevs:properties>
    </wlevs:event-type>
    <wlevs:event-type type-name="PercentTick">
      <wlevs:properties>
        <wlevs:property name="symbol" type="char"/>
        <wlevs:property name="lastPrice" type="double"/>
        <wlevs:property name="percentLastPrice" type="double"/>
        <wlevs:property name="startTimestamp" type="bigint"/>
      </wlevs:properties>
    </wlevs:event-type>
    <wlevs:event-type type-name="Symbols">
      <wlevs:properties>
        <wlevs:property name="symbol" type="char" length="10"/>
      </wlevs:properties>
    </wlevs:event-type>
  </wlevs:event-type-repository>

  <wlevs:adapter id="loadgenAdapter" provider="loadgen">
    <wlevs:instance-property name="port" value="9001"/>
    <wlevs:instance-property name="threadContextTimestamp" value="true"/>
  </wlevs:adapter>

  <!-- definition for cache that holds the symbols -->
  <wlevs:caching-system id="signalgenCachingSystem"/>
  <wlevs:cache id="symbolsCache" key-properties="symbol" value-type="Symbols">
    <wlevs:caching-system ref="signalgenCachingSystem"/>
    <wlevs:cache-loader ref="symbolsCacheLoader"/>
  </wlevs:cache>

  <wlevs:processor id="processor1">
    <wlevs:cache-source ref="symbolsCache"/>
  </wlevs:processor>

  <!-- Streams are just place-holders in this scenario, they just pass-through -->
  <wlevs:channel id="StockTickStream" max-threads="0" max-size="0"
event-type="OracleStockTick">
```

```

        <wlevs:listener ref="processor1"/>
        <wlevs:source ref="loadgenAdapter"/>
    </wlevs:channel>

    <!-- advertise for monitoring -->
    <wlevs:channel id="TrendStream" max-threads="0" max-size="0" advertise="true"
event-type="TrendTick">

        <wlevs:listener ref="outputbean"/>
        <wlevs:source ref="processor1"/>
    </wlevs:channel>

    <wlevs:channel id="PercentStream" max-threads="0" max-size="0" advertise="true"
event-type="PercentTick">

        <wlevs:listener ref="outputbean"/>
        <wlevs:source ref="processor1"/>
    </wlevs:channel>

    <bean id="outputbean" class="oracle.cep.example.signalgen.SignalgenOutputBean">
        <property name="timestampProperty" value="startTimestamp"/>
    </bean>

    <bean id="symbolsCacheLoader" class="oracle.cep.example.signalgen.SymbolsCacheLoader">
        <property name="symbolsFileName" value="applications/MySigGen/symbols.txt"/>
        <property name="eventName" value="Symbols"/>
    </bean>

</beans>

```

In the preceding example:

- The `wlevs:event-type-repository` element creates the event types that are used throughout the application as tuples; in the signal generation application, there are the following events:
 - `TrendTick`: defined in the EPN assembly file.
 - `PercentTick`: defined in the EPN assembly file.
 - `Symbols`: defined in the EPN assembly file.
- The set of `wlevs:adapter`, `wlevs:processor`, `wlevs:channel`, and `wlevs:caching-system` entries set up the event processor network by declaring each component in the network as described in [Figure 3–3](#).

Each component is given a unique ID which can be referenced by other components when they declare their listeners and sources.

- The `wlevs:adapter` element specifies the adapter, for example:

```

<wlevs:adapter id="loadgenAdapter" provider="loadgen">
    <wlevs:instance-property name="port" value="9001"/>
    <wlevs:instance-property name="threadContextTimestamp" value="true"/>
</wlevs:adapter>

```

The `wlevs:adapter` element `provider="loadgen"` attribute specifies that the adapter gets its data from the Oracle CEP load generator utility. The `wlevs:instance-property` child element specifies the port number to which the adapter should listen.

- The `wlevs:processor` element specifies the Oracle CQL processor, for example:

```

<wlevs:processor id="processor1">
    <wlevs:cache-source ref="symbolsCache"/>

```

```
</wlevs:processor>
```

The `listener` attribute, common to all component tags, specifies the component that listens to the processor; in this case, the listener is `symbolsCache`.

- The `wlevs:caching-system` element specifies the Oracle CEP local event cache the application uses to improve performance, for example:

```
<wlevs:caching-system id="signalgenCachingSystem"/>
<wlevs:cache id="symbolsCache" key-properties="symbol"
value-type="Symbols">
  <wlevs:caching-system ref="signalgenCachingSystem"/>
  <wlevs:cache-loader ref="symbolsCacheLoader"/>
</wlevs:cache>
```

For more information on caches, see "Configuring Oracle CEP Caching" in the *Oracle CEP IDE Developer's Guide for Eclipse*.

3.8.6 The Signal Generation Component Configuration Files

The Signal Generation application configures its processor and cache in a component configuration file that [Example 3–8](#) shows.

Example 3–8 Signal Generation Component Configuration File

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- this is here to test that we ignore this file -->
<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>processor1</name>
    <rules>
      <view id="S" schema="symbol price">
        <![CDATA[
          RStream(select T.symbol, T.lastPrice from StockTickStream[now] as T, symbolsCache
as R where T.symbol = R.symbol)
        ]]>
      </view>
      <query id="percent">
        <![CDATA[
          select symbol, lastPrice, percentLastPrice, startTimestamp
          from S MATCH_RECOGNIZE (
            PARTITION BY symbol
            MEASURES
              B.symbol as symbol,
              B.price as lastPrice,
              100*(B.price - A.price)/A.price as percentLastPrice,
              B.ELEMENT_TIME as startTimestamp
            ALL MATCHES
            PATTERN (A B)
            DEFINE
              B AS (100*(B.price - A.price)/A.price > 2.0
                or 100*(B.price - A.price)/A.price < -2.0
                )
            ) as T
        ]]>
      </query>
      <view id="vTrend" schema="symbol lastPrice trendLastPrice">
        <![CDATA[
          select symbol, lastPrice, trendLastPrice
          from S MATCH_RECOGNIZE (
```

```

PARTITION BY symbol
MEASURES
  Z.symbol as symbol,
  Z.price as lastPrice,
  count(A.*)-count(B.*) as trendLastPrice
ALL MATCHES
PATTERN (X (A|B|C) (A|B|C) (A|B|C) (A|B|C) )
SUBSET Z = (A, B, C)
DEFINE
  A AS (A.price > PREV(A.price)),
  B AS (B.price < PREV(B.price)),
  C AS (C.price = PREV(C.price))
) as T
]]>
</view>
<query id="trend">
  <![CDATA[
    select symbol, lastPrice, trendLastPrice, ELEMENT_TIME as startTimestamp from vTrend
  where trendLastPrice > 2
  ]]>
</query>
</rules>
</processor>
<channel>
  <name>TrendStream</name>
  <selector>trend</selector>
</channel>
<channel>
  <name>PercentStream</name>
  <selector>percent</selector>
</channel>
</nl:config>

```

If your application contains multiple processors, adapters or channels, you can either declare them all in a single configuration file, or create separate configuration files for each component; the method you chose depends on which you find easier to manage.

For each component you configure, you must add the name child element to explicitly declare the specific component to which you are referring. The value of the name element must correspond to the component's unique identifier of its declaration in the EPN assembly file.

For example, assume a processor is declared in the EPN assembly file as follows:

```
<wlevs:processor id="processor1" ...>
```

Then its corresponding XML configuration would be as follows:

```

<processor>
  <name>processor1</name>
  ...
</processor>

```

The Signal Generation example uses a single configuration file for one processor with the name `processor1` and one cache with the name `symbolsCache`. These names correspond with the declaration of the components in the EPN assembly file.

The `processor` element configures the processor component. The most important part of the processor configuration is the declaration of the set of Oracle Continuous Query Language (Oracle CQL) rules that this processor executes; these rules select the set of events that are eventually passed to the application business object. Each rule is declared with a `query` or `relation` element using an XML `<![CDATA[...]]>` section; all `query` and `relation` elements are grouped together within a single `rules` element. You can define as many rules as you want for a particular processor.

The Signal Generation application has the following rules:

```

<rules>
  <view id="S" schema="symbol price">
    <![CDATA[
      RStream(select T.symbol, T.lastPrice from StockTickStream[now] as T, symbolsCache
as R where T.symbol = R.symbol)
    ]]>
  </view>
  <query id="percent">
    <![CDATA[
      select symbol, lastPrice, percentLastPrice, startTimestamp
      from S MATCH_RECOGNIZE (
        PARTITION BY symbol
        MEASURES
          B.symbol as symbol,
          B.price as lastPrice,
          100*(B.price - A.price)/A.price as percentLastPrice,
          B.ELEMENT_TIME as startTimestamp
        ALL MATCHES
        PATTERN (A B)
        DEFINE
          B AS (100*(B.price - A.price)/A.price > 2.0
              or 100*(B.price - A.price)/A.price < -2.0
              )
        ) as T
    ]]>
  </query>
  <view id="vTrend" schema="symbol lastPrice trendLastPrice">
    <![CDATA[
      select symbol, lastPrice, trendLastPrice
      from S MATCH_RECOGNIZE (
        PARTITION BY symbol
        MEASURES
          Z.symbol as symbol,
          Z.price as lastPrice,
          count(A.*)-count(B.*) as trendLastPrice
        ALL MATCHES
        PATTERN (X (A|B|C) (A|B|C) (A|B|C) (A|B|C) )
        SUBSET Z = (A, B, C)
        DEFINE
          A AS (A.price > PREV(A.price)),
          B AS (B.price < PREV(B.price)),
          C AS (C.price = PREV(C.price))
        ) as T
    ]]>
  </view>
  <query id="trend">
    <![CDATA[
      select symbol, lastPrice, trendLastPrice, ELEMENT_TIME as startTimestamp from vTrend
      where trendLastPrice > 2
    ]]>
  </query>
</rules>

```

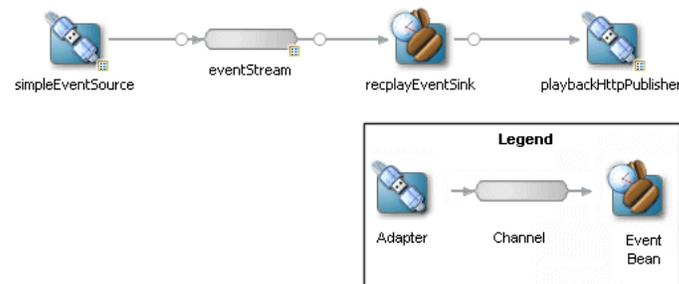
For more information, see *Oracle CEP CQL Language Reference*.

3.9 Event Record and Playback Example

The record and playback example shows how to configure a component to record events to an event store and then configure another component in the network to playback events from the store. The example uses the Oracle CEP-provided database server, Apache Derby, to store the events. The example also shows how to configure a publishing HTTP pub-sub adapter as a node in the event processing network.

Figure 3–5 shows the event record and playback example Event Processing Network (EPN). The EPN contains the components that make up the application and how they fit together.

Figure 3–5 The Event Record and Playback Example Event Processing Network



The application contains four components in its event processing network:

- `simpleEventSource`: an adapter that generates simple events for purposes of the example. This component has been configured to record events, as shown in the graphic.

The configuration source for this adapter is:

```

<adapter>
  <name>simpleEventSource</name>
  <record-parameters>
    ...
  </record-parameters>
</adapter>
  
```

- `eventStream`: a channel that connects the `simpleEventSource` adapter and `replayEventSink` event bean. This component has been configured to playback events.

The configuration source for this channel is:

```

<channel>
  <name>eventStream</name>
  <playback-parameters>
    ...
  </playback-parameters>
  ...
</channel>
  
```

- `replayEventSink`: an event bean that acts as a sink for the events generated by the adapter.
- `playbackHttpPublisher`: a publishing HTTP pub-sub adapter that listens to the `replayEventSink` event bean and publishes to a channel called `/playbackchannel` of the Oracle CEP HTTP Pub-Sub server.

3.9.1 Running the Event Record/Playback Example

The `replay_domain` domain contains a single application: the record and playback sample application. To run this application, you first start an instance of Oracle CEP in the domain, as described in the following procedure.

The procedure then shows you how to use Oracle CEP Visualizer to start the recording and playback of events at the `simpleEventSource` and `eventStream` components,

respectively. Finally, the procedure shows you how to use Oracle CEP Visualizer to view the stream of events being published to a channel by the `playbackHttpPublisher` adapter.

To run the event record/playback example:

1. Open a command window and change to the default server directory of the `recplay_domain` directory, located in `ORACLE_CEP_HOME\ocep_11.1\samples\domains\recplay_domain\defaultserver`, where `ORACLE_CEP_HOME` refers to the main Oracle CEP installation directory, such as `d:\oracle_cep`.
2. Set your development environment, as described in [Section 3.5, "Setting Your Development Environment."](#)
3. Start Oracle CEP by executing the appropriate script with the correct command line arguments:

a. On Windows:

- * If you are using the JRockit JDK included in Oracle JRockit Real Time 3.0, enable the deterministic garbage collector by passing the `-dgc` parameter to the command:

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

- * If you are not using the JRockit JDK included in Oracle JRockit Real Time 3.0:

```
prompt> startwlevs.cmd
```

b. On UNIX:

- * If you are using the JRockit JDK included in Oracle JRockit Real Time 3.0, enable the deterministic garbage collector by passing the `-dgc` parameter to the command:

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

- * If you are not using the JRockit JDK included in Oracle JRockit Real Time 3.0:

```
prompt> startwlevs.sh
```

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

```
SimpleEvent created at: 14:33:40.441
```

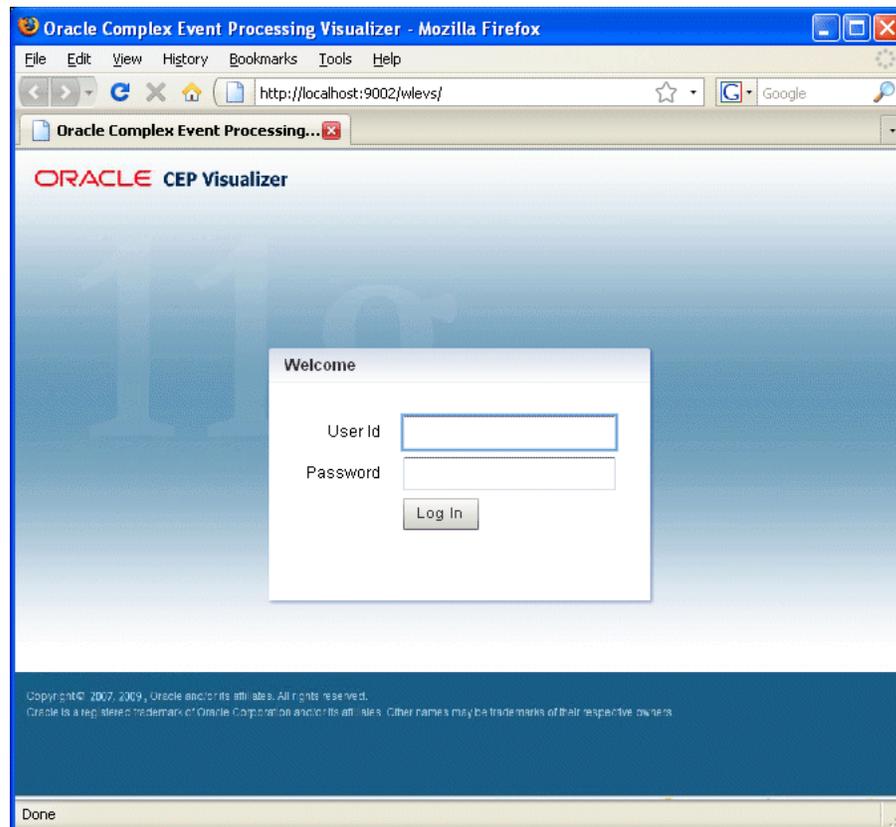
This message indicates that the Oracle CEP server started correctly and that the `simpleEventSource` component is creating events.

4. Invoke the following URL in your browser:

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

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

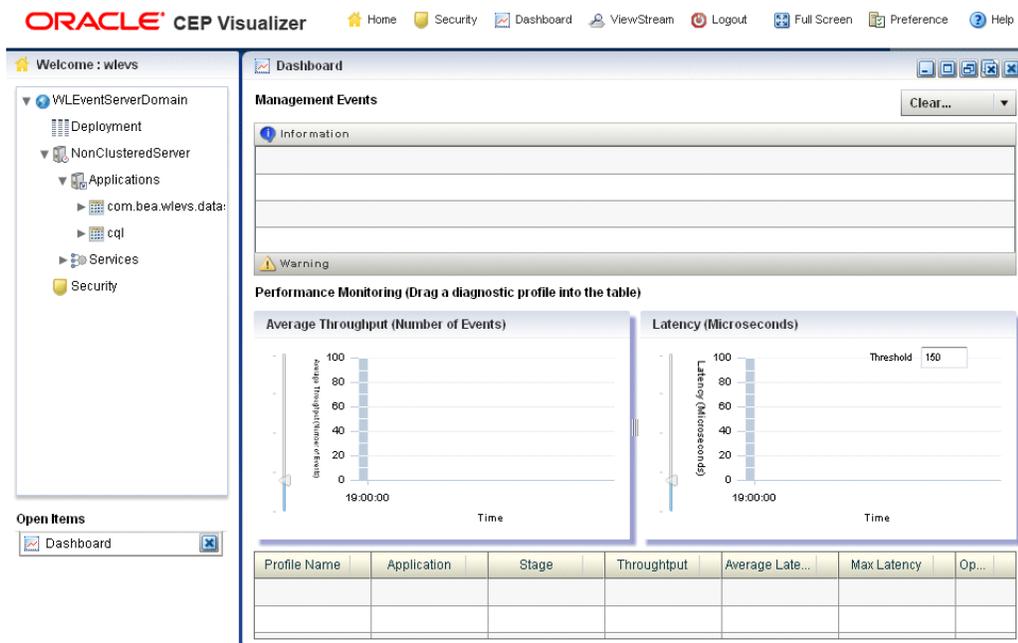
The Logon screen appears as [Figure 3-6](#) shows.

Figure 3–6 Oracle CEP Visualizer Logon Screen

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

The Oracle CEP Visualizer dashboard appears as [Figure 3–40](#) shows.

Figure 3–7 Oracle CEP Visualizer Dashboard



For more information about the Oracle CEP Visualizer user interface, see "Understanding the Oracle CEP Visualizer User Interface" in the *Oracle CEP Visualizer User's Guide*.

- In the left pane, select **WLEventServerDomain > NonClusteredServer > Applications > replay > Stages > simpleEventSource**.
- In the right pane, select the **Record** tab as shown in [Figure 3–8](#).

Figure 3–8 Event Record Tab

The **DataSet Name** field contains the value of the `record-parameters` child element `dataset-name` element from the `simpleEventSource` adapter application configuration file `ORACLE-CEP-HOME\ocp_11.1\samples\domains\replay_domain\defaultserver\applications\replay\config.xml` as [Example 3–9](#) shows.

Example 3–9 *replay* Application Configuration File *config.xml*: adapter Element

```
<adapter>
  <name>simpleEventSource</name>
  <record-parameters>
    <dataset-name>replay_sample</dataset-name>
    <event-type-list>
      <event-type>SimpleEvent</event-type>
    </event-type-list>
    <provider-name>test-rdbms-provider</provider-name>
    <batch-size>1</batch-size>
    <batch-time-out>10</batch-time-out>
  </record-parameters>
</adapter>
```

The **Provider Name** contains the value of the `rdbms-event-store-provider` child element name which corresponds to the `data-source` child element name as [Example 3–10](#) shows.

Example 3–10 *replay* Oracle CEP Server Configuration File *config.xml*: *data-source* and *rdbms-event-store-provider* Elements

```
<data-source>
```

```
<name>derby1</name>
<connection-pool-params>
  <initial-capacity>15</initial-capacity>
  <max-capacity>50</max-capacity>
</connection-pool-params>
<driver-params>
  <url>jdbc:derby:dbtest1;create=true</url>
  <driver-name>org.apache.derby.jdbc.EmbeddedDriver</driver-name>
</driver-params>
</data-source>
<rdbms-event-store-provider>
  <name>test-rdbms-provider</name>
  <data-source-name>derby1</data-source-name>
</rdbms-event-store-provider>
```

8. At the bottom of the Record tab, click **Start**.

An Alert dialog appears as shown in [Figure 3–9](#).

Figure 3–9 Start Recording Alert Dialog



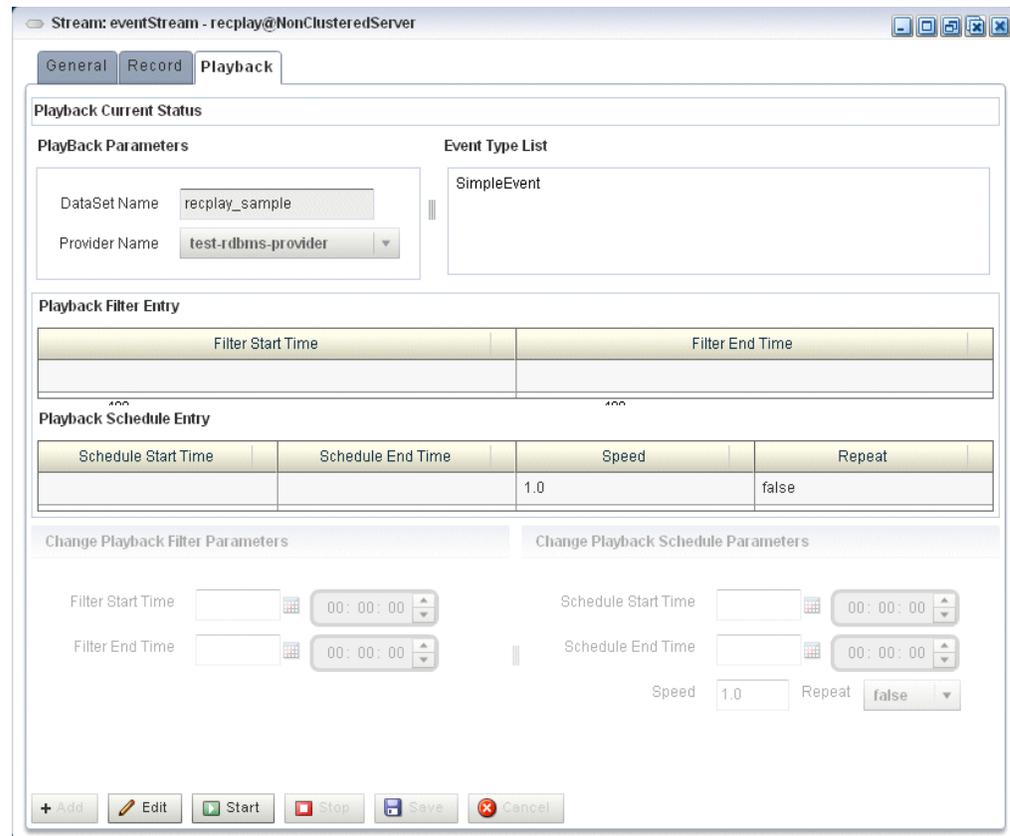
9. Click **OK**.

The Current Status field reads **Recording...**

As soon as you click **OK**, events start to flow out of the `simpleEventSource` component and are stored in the configured database.

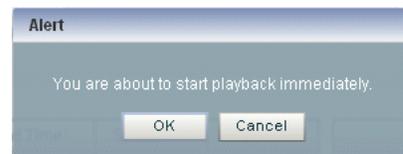
You can further configure when events are recorded using the **Start Recording** and **Stop Recording** fields.

10. In the left pane, select **eventStream**.
11. In the right pane, select the **Playback** tab as shown in [Figure 3–10](#).

Figure 3–10 Event Playback Tab

12. At the bottom of the tab, click **Start**.

An Alert dialog appears as shown in [Figure 3–11](#).

Figure 3–11 Start Playback Alert Dialog

13. Click **OK**.

The Current Status field reads **Playing...**

As soon as you click **OK**, events that had been recorded by the `simpleEventSource` component are now played back to the `simpleStream` component.

You should see the following messages being printed to the command window from which you started Oracle CEP server to indicate that both original events and playback events are streaming through the EPN:

```
SimpleEvent created at: 14:33:11.501
Played back: Original time=14:15:23.141 Playback time=14:33:11.657
```

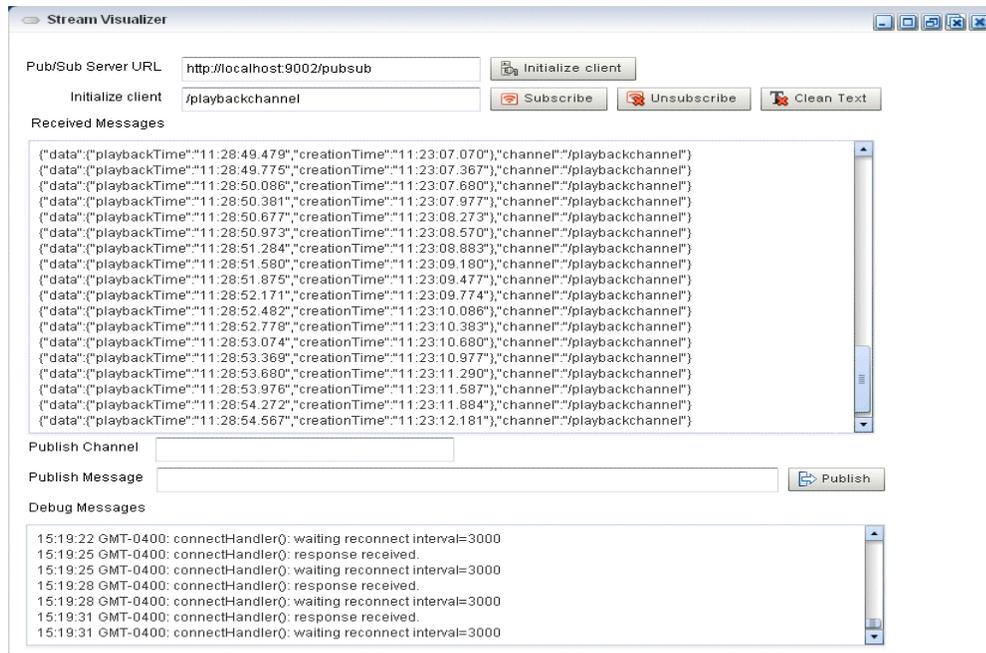
You can further configure the playback parameters, such as the recorded time period for which you want playback events and the speed that they are played

back, by updating the appropriate field and clicking **Change Parameters**. You must restart the playback after changing any playback parameters.

14. To view the events that the `playbackHttpPublisher` adapter is publishing to a channel, follow these steps:
 - a. In the top pane, select **Viewstream**.

The Viewstream window appears as shown in [Figure 3–12](#).

Figure 3–12 Stream Visualizer



- b. In the right pane, click **Initialize Client**.
- c. In the Subscribe Channel text box, enter `/playbackchannel`.
- d. Click **Subscribe**.

The **Received Messages** text box displays the played back event details. The played back events show the time at which the event was created and the time at which it was played back.

3.9.2 Building and Deploying the Event Record/Playback Example from the Source Directory

The record and playback sample source directory contains the Java source, along with other required resources, such as configuration XML file and EPN assembly file that make up the application. The `build.xml` Ant file contains targets to build and deploy the application to the `signalgeneration_domain` domain, as described in [Section 3.9.3, "Description of the Ant Targets to Build the Record and Playback Example."](#)

To build and deploy the event record/playback example from the source directory:

1. If the record/playback Oracle CEP instance is not already running, follow the procedure in [Section 3.9.1, "Running the Event Record/Playback Example"](#) to start

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

2. Open a new command window and change to the record/playback source directory, located in `ORACLE_CEP_HOME\ocep_11.1\samples\source\applications\recplay`, where `ORACLE_CEP_HOME` refers to the main Oracle CEP installation directory, such as `d:\oracle_cep`.

```
prompt> cd d:\oracle_cep\ocep_11.1\samples\source\applications\recplay
```
3. Set your development environment, as described in [Section 3.5, "Setting Your Development Environment."](#)
4. Execute the `all` Ant target to compile and create the application JAR file:

```
prompt> ant all
```
5. Execute the `deploy` Ant target to deploy the application JAR file to the `ORACLE_CEP_HOME\ocep_11.1\samples\domains\recplay_domain\defaultserver\applications\recplay` directory:

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

Caution: This target overwrites the existing event record/playback application JAR file in the domain directory.

After an application redeploy message, you should see the following message printed to the output about every second:

```
SimpleEvent created at: 14:33:40.441
```

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

6. Follow the instructions in [Section 3.9.1, "Running the Event Record/Playback Example,"](#) starting at step 4, to invoke Oracle CEP Visualizer and start recording and playing back events.

3.9.3 Description of the Ant Targets to Build the Record and Playback Example

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

- `clean`—This target removes the `dist` and output working directories under the current directory.
- `all`—This target cleans, compiles, and jars up the application into a file called `com.bea.wlevs.example.recplay_3.0.0.0.jar`, and places the generated JAR file into a `dist` directory below the current directory.
- `deploy`—This target deploys the JAR file to Oracle CEP using the Deployer utility. For more information, see "Deployer Command-Line Reference" in the *Oracle CEP Administrator's Guide*.

3.9.4 Implementation of the Record and Playback Example

The implementation of the signal generation example generally follows "Creating Oracle CEP Applications: Typical Steps" in the *Oracle CEP IDE Developer's Guide for Eclipse*.

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

All the files of the example are located relative to the `ORACLE_CEP_HOME\ocep_11.1\samples\source\applications\recplay` directory, where `ORACLE_CEP_HOME` refers to the main Oracle CEP installation directory such as `c:\oracle_cep`. Oracle recommends that you use this example directory setup in your own environment, although it is obviously not required.

The files used by the record and playback example include:

- An EPN assembly file that describes each component in the application and how all the components are connected together as shown in [Figure 3-5](#).

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

- Java source file for the `simpleEventSource` adapter.

In the example, the file is called `SimpleEventSource.java` and is located in the `src/com/bea/wlevs/adapter/example/recplay` directory.

For a detailed description of this file and how to program the adapter Java files in general, see "Creating Custom Adapters and Event Beans" in the *Oracle CEP IDE Developer's Guide for Eclipse*.

- Java source file that describes the `PlayedBackEvent` and `SimpleEvent` event types. The `SimpleEvent` event type is the one originally generated by the adapter, but the `PlayedBackEvent` event type is used for the events that are played back after having been recorded. The `PlayedBackEvents` look almost exactly the same as `SimpleEvent` except they have an extra field, the time the event was recorded.

In the example, the two events are called `SimpleEvent.java` and `PlayedBackEvent.java` and are located in the `src/com/bea/wlevs/event/example/recplay` directory.

For a detailed description of this file, as well as general information about programming event types, see "Creating the Event Types" in the *Oracle CEP IDE Developer's Guide for Eclipse*.

- A Java file that implements the `recplayEventSink` event bean of the application, which is an event sink that receives both realtime events from the `simpleEventSource` adapter as well as playback events.

In the example, the file is called `RecplayEventSink.java` and is located in the `src/com/bea/wlevs/example/recplay` directory.

For a detailed description of this file and how to program the adapter Java files in general, see "Creating Custom Adapters and Event Beans" in the *Oracle CEP IDE Developer's Guide for Eclipse*.

- An XML file that configures the `simpleEventSource` adapter and `eventStream` channel components. The adapter includes a `<record-parameters>` element that specifies that the component will record

events to the event store; similarly, the channel includes a `<playback-parameters>` element that specifies that it receives playback events.

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

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

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

For more information about creating this file, as well as a description of creating the OSGi bundle that you deploy to Oracle CEP, see "Overview of Application Assembly and Deployment" in the *Oracle CEP IDE Developer's Guide for Eclipse*.

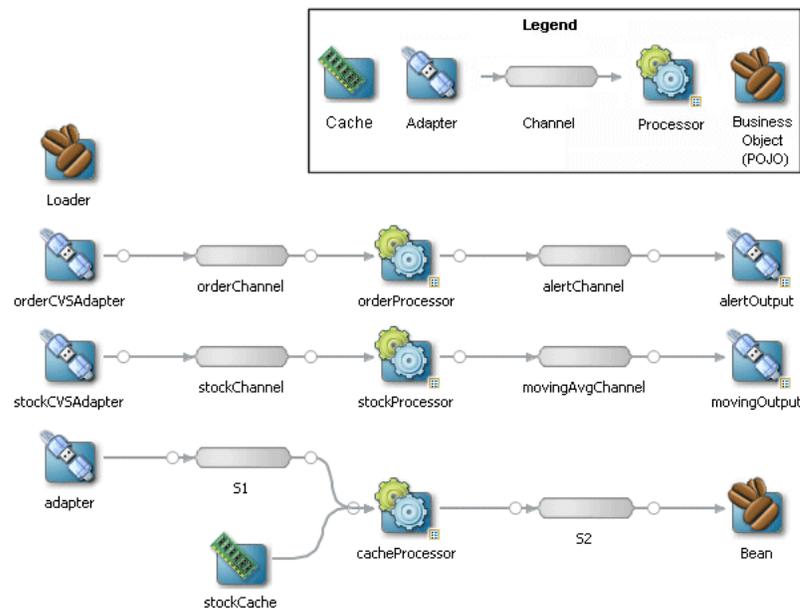
The record/playback example uses a `build.xml` Ant file to compile, assemble, and deploy the OSGi bundle; see [Section 3.9.2, "Building and Deploying the Event Record/Playback Example from the Source Directory"](#) for a description of this `build.xml` file if you also use Ant in your development environment.

3.10 Oracle Continuous Query Language (CQL) Example

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

[Figure 3-13](#) shows the CQL example Event Processing Network (EPN). The EPN contains the components that make up the application and how they fit together.

Figure 3-13 The CQL Example Event Processing Network



The application contains three separate event paths in its EPN:

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

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

For more information on how to construct the query that the `cqlProc` processor executes, see [Section 3.10.4.1, "Creating the Missing Event Query"](#).

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

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

For more information on how to construct the query that the `cqlProc` processor executes, see [Section 3.10.4.2, "Creating the Moving Average Query"](#).

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

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

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

3.10.1 Running the CQL Example

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

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

To run the CQL example:

1. Open a command window and change to the default server directory of the CQL domain directory, located in `ORACLE_CEP_HOME\ocep_11.1\samples\domains\cql_domain\defaultserver`, where `ORACLE_CEP_HOME` refers to the main Oracle CEP installation directory, such as `d:\oracle_cep`.

```
prompt> cd d:\oracle_cep\ocep_11.1\samples\domains\cql_domain\defaultserver
```
2. Set your development environment, as described in [Section 3.5, "Setting Your Development Environment."](#)
3. Start Oracle CEP by executing the appropriate script with the correct command line arguments:
 - a. On Windows:
 - * If you are using the JRockit JDK included in Oracle JRockit Real Time 3.0, enable the deterministic garbage collector by passing the `-dgc` parameter to the command:

```
prompt> startwlevs.cmd -dgc
```
 - * If you are not using the JRockit JDK included in Oracle JRockit Real Time 3.0:

```
prompt> startwlevs.cmd
```
 - b. On UNIX:

- * If you are using the JRockit JDK included in Oracle JRockit Real Time 3.0, enable the deterministic garbage collector by passing the `-dgc` parameter to the command:

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

- * If you are not using the JRockit JDK included in Oracle JRockit Real Time 3.0:

```
prompt> startwlevs.sh
```

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

4. To simulate the data feed for the missing event query, open a new command window and set your environment as described in [Section 3.5, "Setting Your Development Environment."](#)
5. 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`.

6. Run the load generator using the `orderData.prop` properties file:

- a. On Windows:

```
prompt> runloadgen.cmd orderData.prop
```

- b. On UNIX:

```
prompt> runloadgen.sh orderData.prop
```

7. To simulate the data feed for the moving average query, open a new command window and set your environment as described in [Section 3.5, "Setting Your Development Environment."](#)
8. 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`.

9. Run the load generator using the `stockData.prop` properties file:

- a. On Windows:

```
prompt> runloadgen.cmd stockData.prop
```

- b. On UNIX:

```
prompt> runloadgen.sh stockData.prop
```

10. To simulate the data feed for the cache query, you only need to run the example.

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

3.10.2 Building and Deploying the CQL Example

The CQL sample source directory contains the Java source, along with other required resources such as configuration XML files, that make up the CQL application. The `build.xml` Ant file contains targets to build and deploy the application to the `cql_domain` domain, as described in [Section 3.10.3, "Description of the Ant Targets to Build the CQL Example."](#)

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

1. If the CQL Oracle CEP instance is not already running, follow the procedure in [Section 3.10.1, "Running the CQL Example"](#) to start the server.

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

2. Open a new command window and change to the CQL source directory, located in `ORACLE_CEP_HOME\ocep_11.1\samples\source\applications\cql`, where `ORACLE_CEP_HOME` refers to the main Oracle CEP installation directory, such as `d:\oracle_cep`.

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

3. Set your development environment, as described in [Section 3.5, "Setting Your Development Environment."](#)

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

```
prompt> ant all
```

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

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

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

6. If the load generators required by the CQL application are not running, start them as described in [Section 3.10.1, "Running the CQL Example."](#)

3.10.3 Description of the Ant Targets to Build the CQL Example

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

- `clean`—This target removes the `dist` and output working directories under the current directory.
- `all`—This target cleans, compiles, and jars up the application into a file called `com.bea.wlevs.example.cql_3.0.0.0.jar`, and places the generated JAR file into a `dist` directory below the current directory.
- `deploy`—This target deploys the JAR file to Oracle CEP using the Deployer utility.

For more information, see "Deployer Command-Line Reference" in the *Oracle CEP Administrator's Guide*.

3.10.4 Implementation of the CQL Example

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

- [Section 3.10.4.1, "Creating the Missing Event Query"](#)
- [Section 3.10.4.2, "Creating the Moving Average Query"](#)

3.10.4.1 Creating the Missing Event Query

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

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

In a valid scenario, you see events in the order that [Table 3–1](#) lists:

Table 3–1 Valid Order Workflow

Event Type	Description
C	Customer order
A	Approval
S	Shipment

However, it is an error if an order is shipped without an approval event as [Table 3–2](#) lists:

Table 3–2 Invalid Order Workflow

Event Type	Description
C	Customer order
S	Shipment

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

- ["To create the missing event query:"](#) on page 3-51
- ["To test the missing event query:"](#) on page 3-74

To create the missing event query:

1. If the CQL Oracle CEP instance is not already running, follow the procedure in [Section 3.10.1, "Running the CQL Example"](#) to start the server.

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

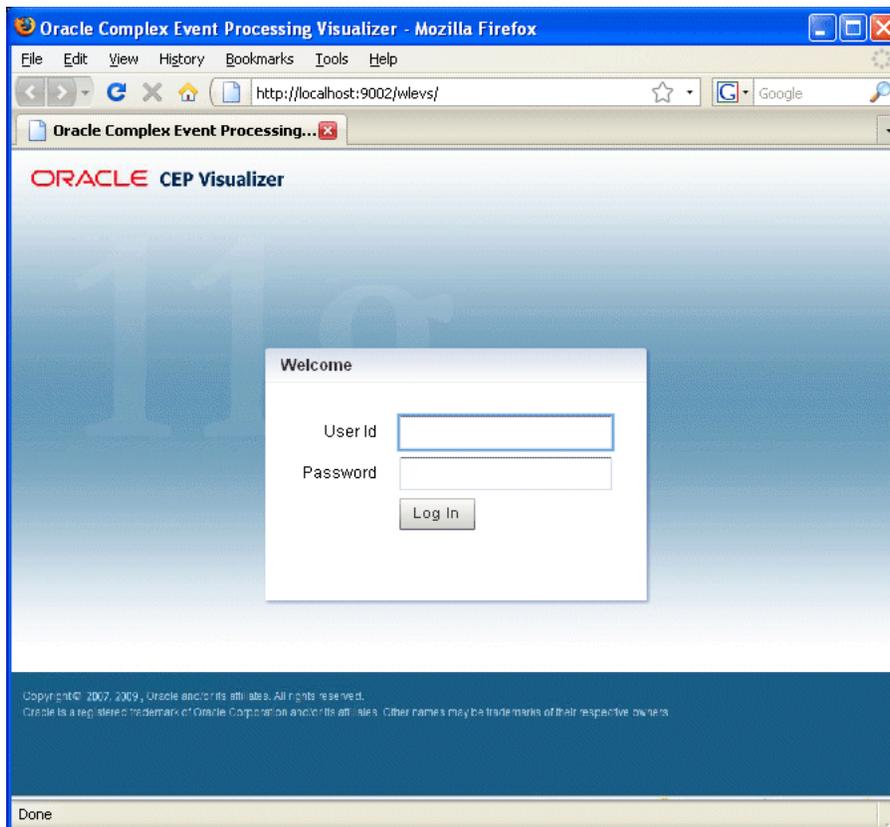
2. Invoke the following URL in your browser:

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

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

The Logon screen appears as [Figure 3–14](#) shows.

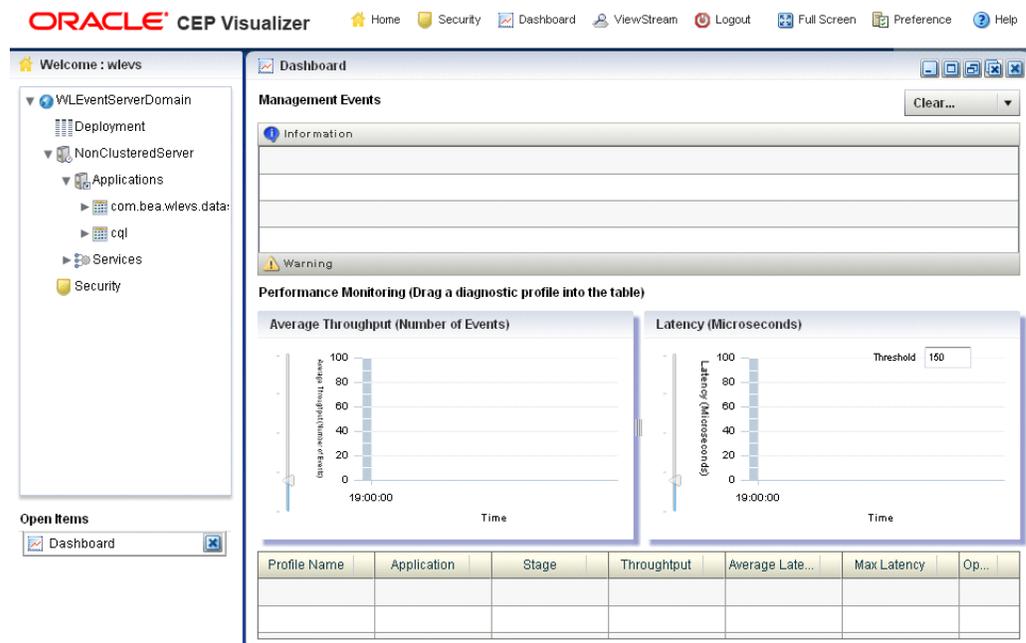
Figure 3–14 Oracle CEP Visualizer Logon Screen



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

The Oracle CEP Visualizer dashboard appears as [Figure 3–40](#) shows.

Figure 3–15 Oracle CEP Visualizer Dashboard

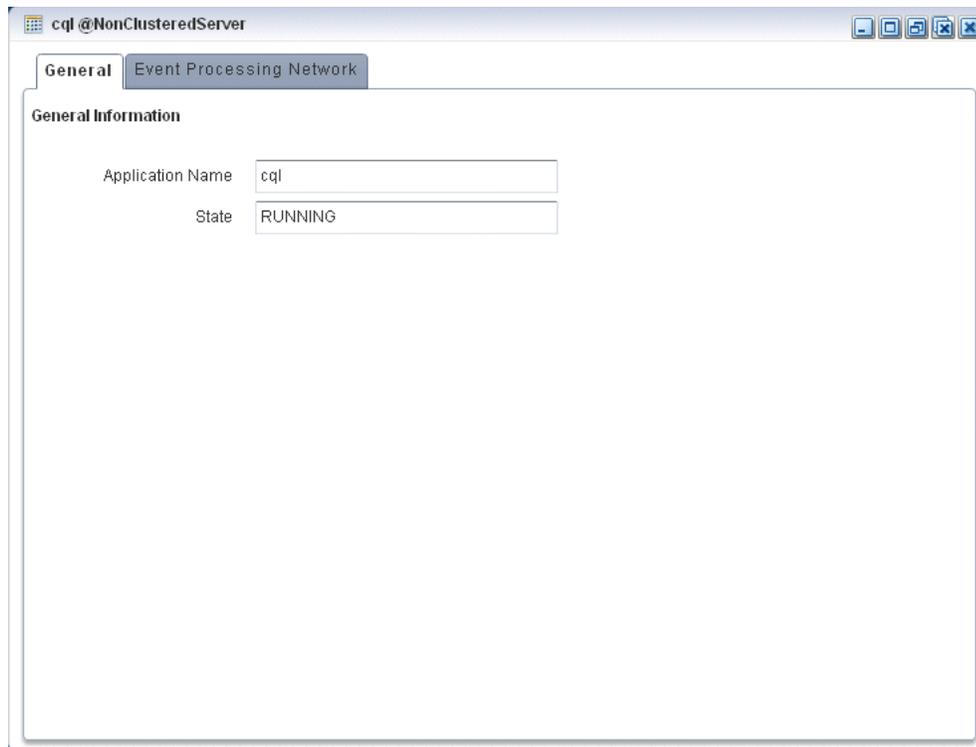


For more information about the Oracle CEP Visualizer user interface, see "Understanding the Oracle CEP Visualizer User Interface" in the *Oracle CEP Visualizer User's Guide*.

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

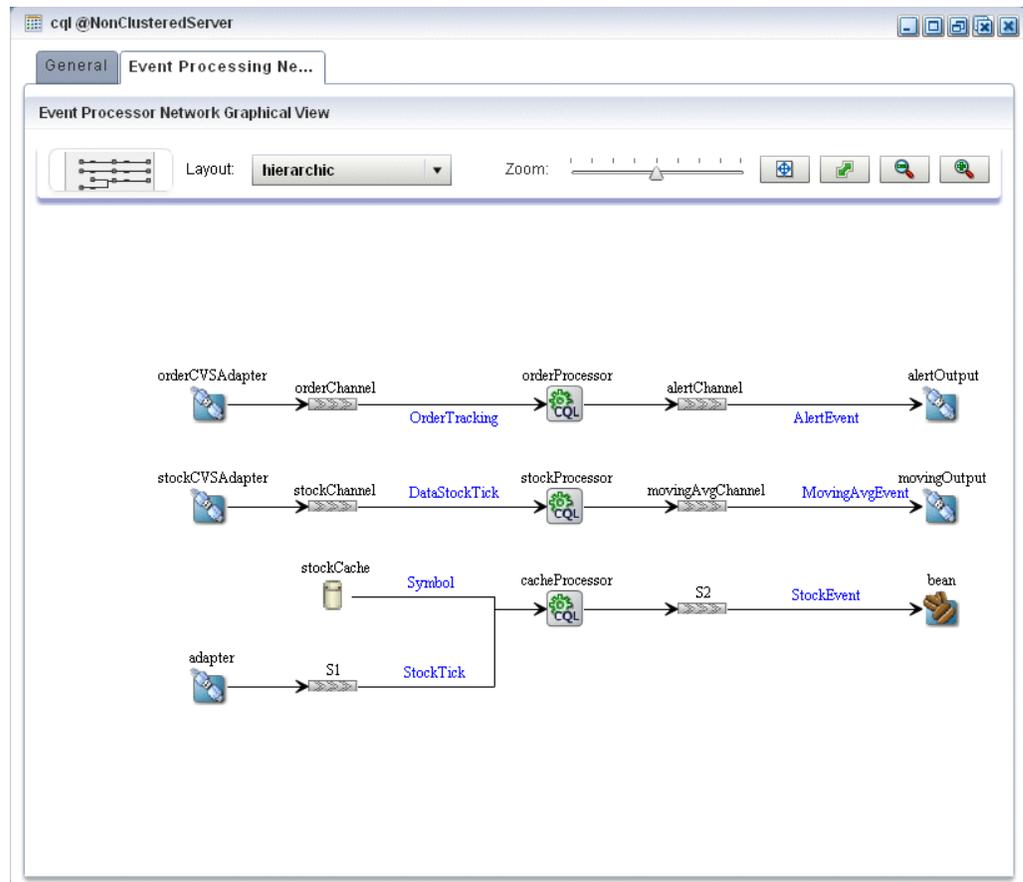
The CQL application screen appears as [Figure 3–41](#) shows.

Figure 3–16 CQL Application Screen: General Tab



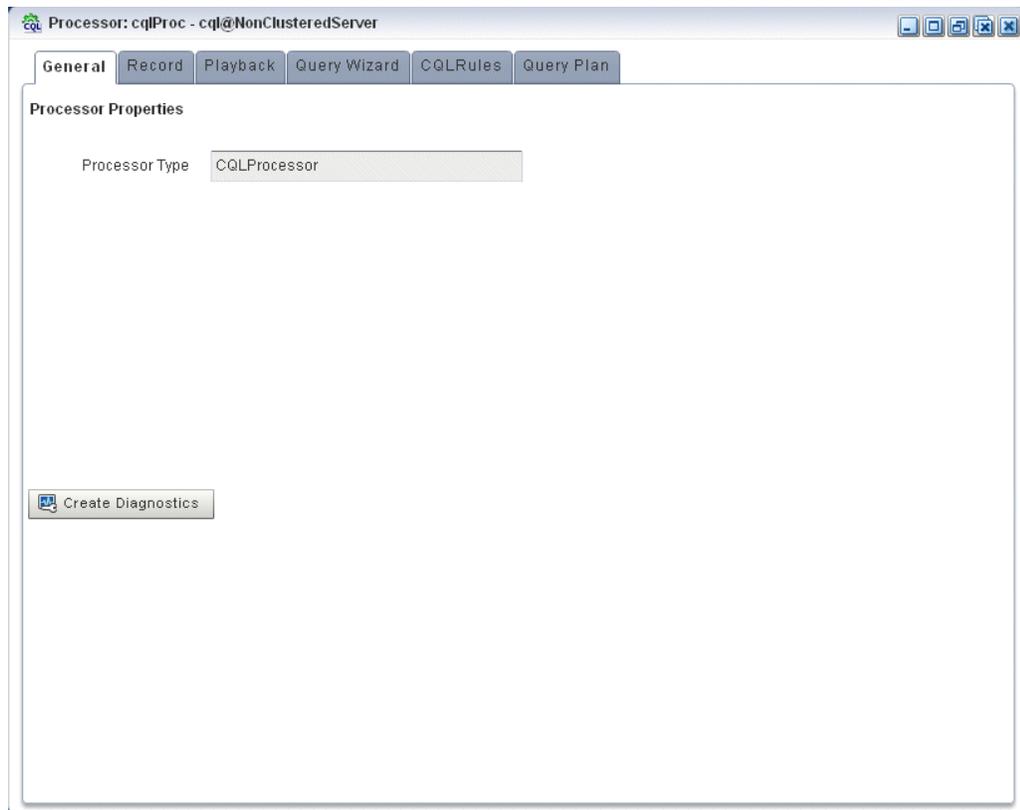
6. Select the **Event Processing Network** tab.

The Event Processing Network screen appears as [Figure 3–42](#) shows.

Figure 3–17 CQL Application: Event Processing Network Tab

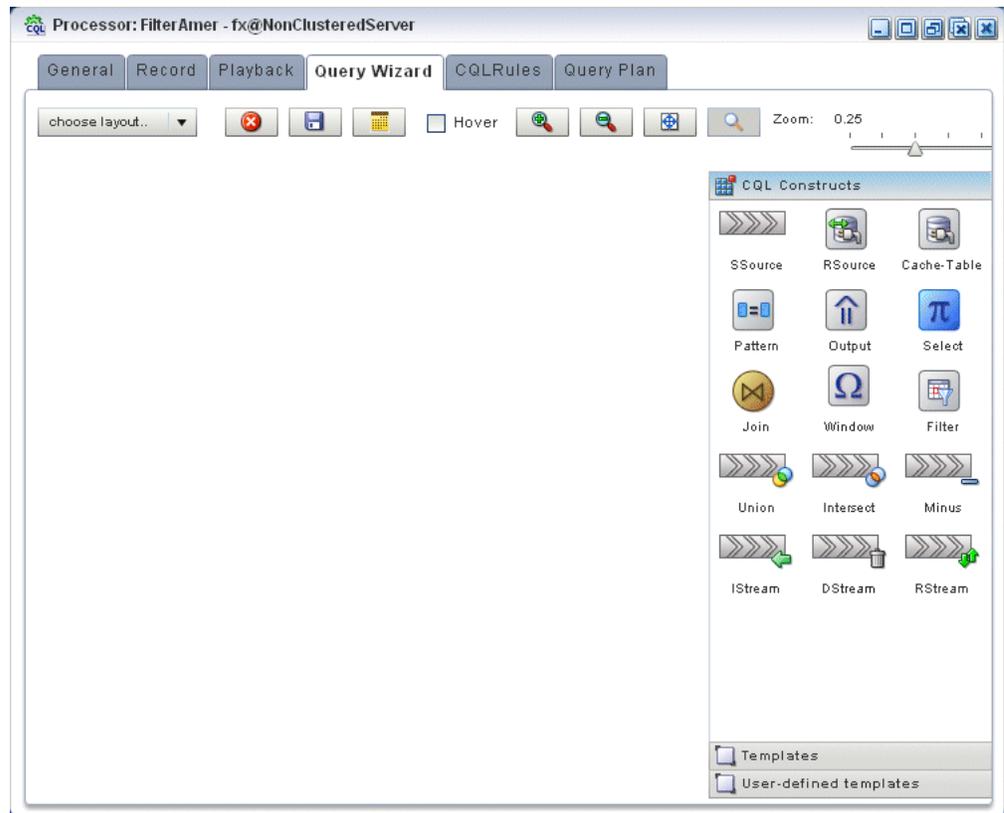
7. Double-click the **orderProcessor** Oracle CQL processor icon.
The Oracle CQL processor screen appears as [Figure 3–43](#) shows.

Figure 3–18 Oracle CQL Processor: General Tab



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

The Query Wizard screen appears as [Figure 3–44](#) shows.

Figure 3–19 Oracle CQL Processor: Query Wizard Tab

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

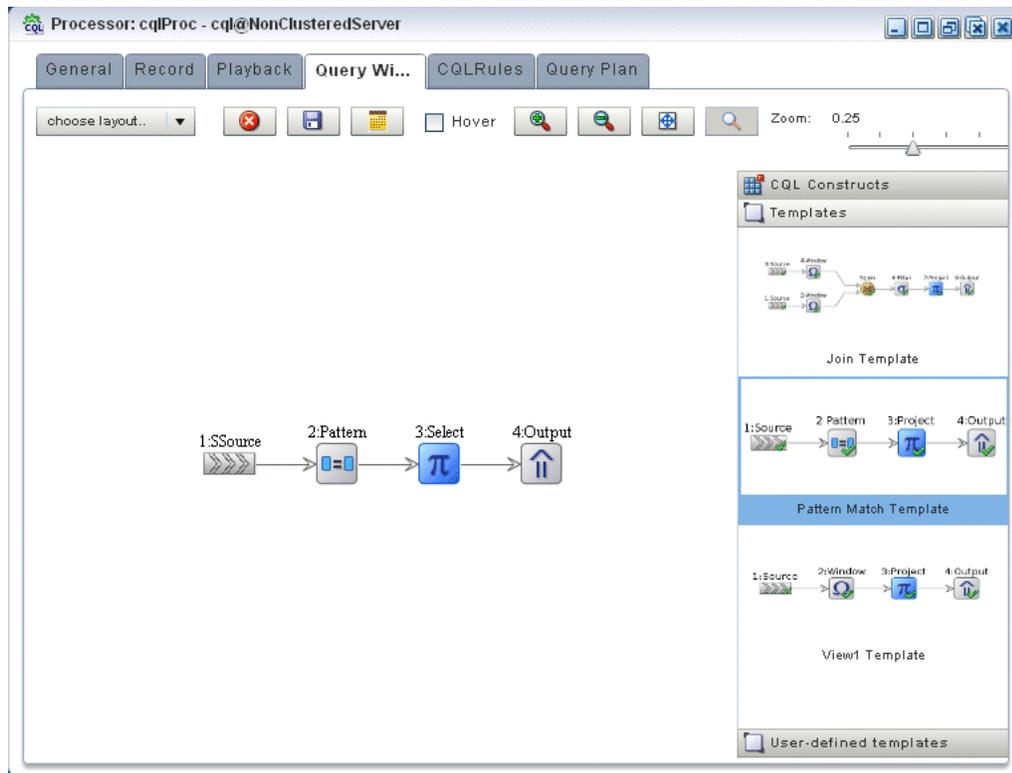
In this procedure, you are going to create an Oracle CQL query from a template.

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

9. Click the Templates tab.

The Templates tab appears as [Figure 3–20](#) shows.

Figure 3–20 Template Tab



10. Click and drag the **Pattern Match Template** from the Templates palette and drop it anywhere in the Query Wizard canvas as [Figure 3–20](#).

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

The SSource configuration screen appears as [Figure 3–46](#) shows.

Figure 3–21 SSource Configuration Dialog

Stream [ID: 1]

Type Stream View

orderChannel AS

Source Properties

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

Generated CQL Statement

```
SELECT * FROM orderChannel
```

Help Validate Save Cancel

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

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

The Pattern configuration screen appears as [Figure 3–22](#) shows.

Figure 3–22 Pattern Configuration Dialog: Pattern Tab



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

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

CustOrder NoApproval*? Shipment

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

Table 3–3 MATCH_RECOGNIZE Pattern Quantifiers

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

For more information, see:

- "PATTERN Condition" in the *Oracle CEP CQL Language Reference*
- "MATCH_RECOGNIZE Condition" in the *Oracle CEP CQL Language Reference*

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

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

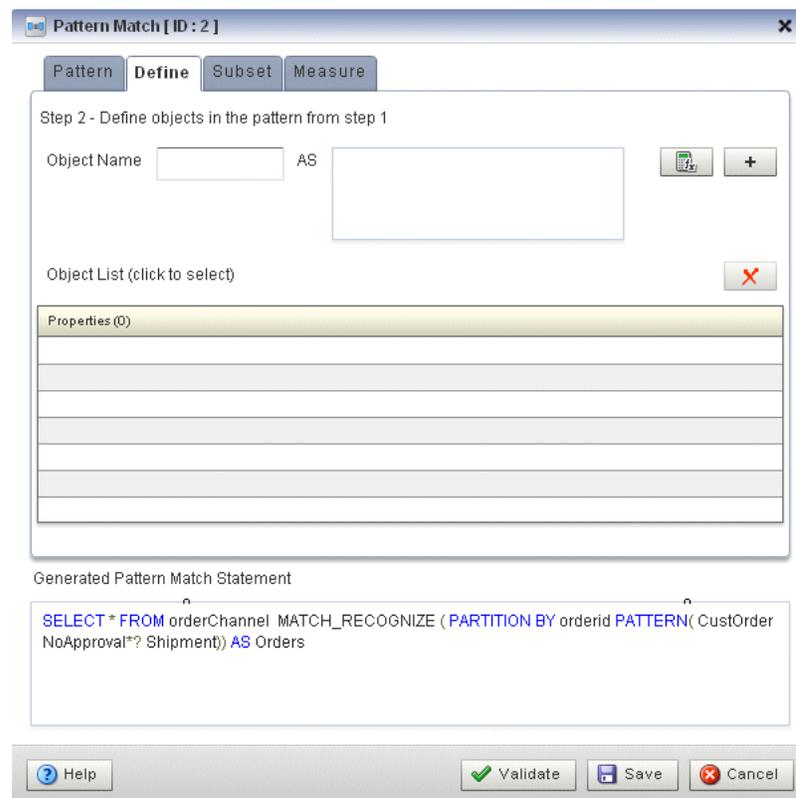
18. Enter `Orders` in the `Alias` field.

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

19. Click the `Define` tab.

The Define tab appears as [Figure 3–23](#) shows.

Figure 3–23 Pattern Configuration Dialog: Define Tab



We will now define each of the conditions named in the pattern clause as [Table 3–4](#) lists:

Table 3–4 Condition Definitions

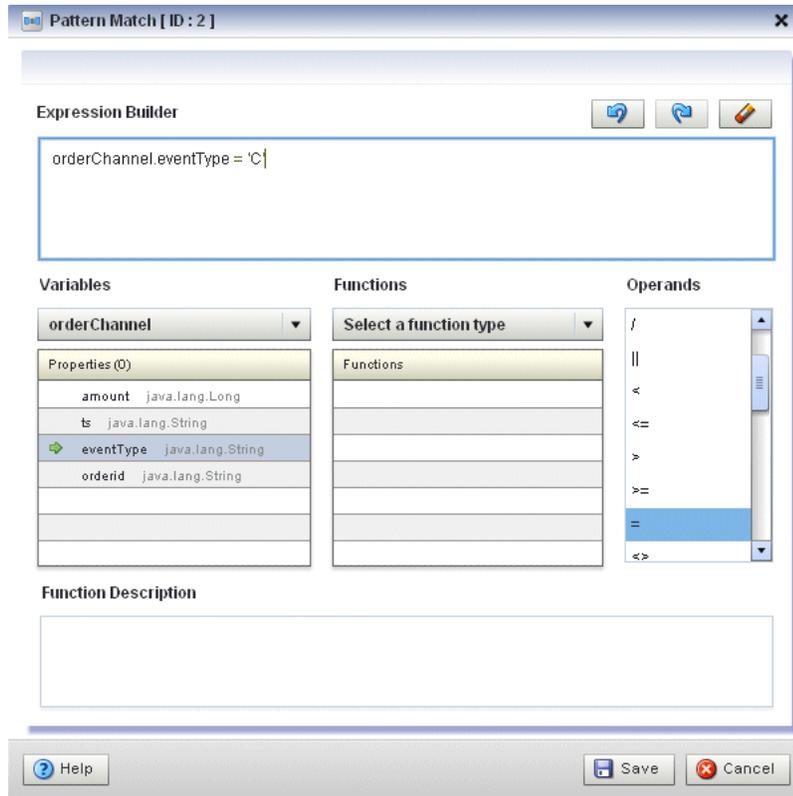
Condition Name	Definition
CustOrder	<code>orderChannel.eventType = 'C'</code>
NoApproval	<code>NOT(orderChannel.eventType = 'A')</code>
Shipment	<code>orderChannel.eventType = 'C'</code>

20. Enter `CustOrder` in the `Object Name` field.

21. Click the Expression Builder button and configure the Expression Builder as follows (see [Figure 3–24](#)):

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

Figure 3–24 Expression Builder: CustOrder



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

The condition definition is added to the Object List as [Figure 3–25](#) shows.

Figure 3–25 Pattern Configuration Dialog: Define Tab With CustOrder Condition

Pattern Match [ID : 2]

Pattern Define Subset Measure

Step 2 - Define objects in the pattern from step 1

Object Name AS

Object List (click to select)

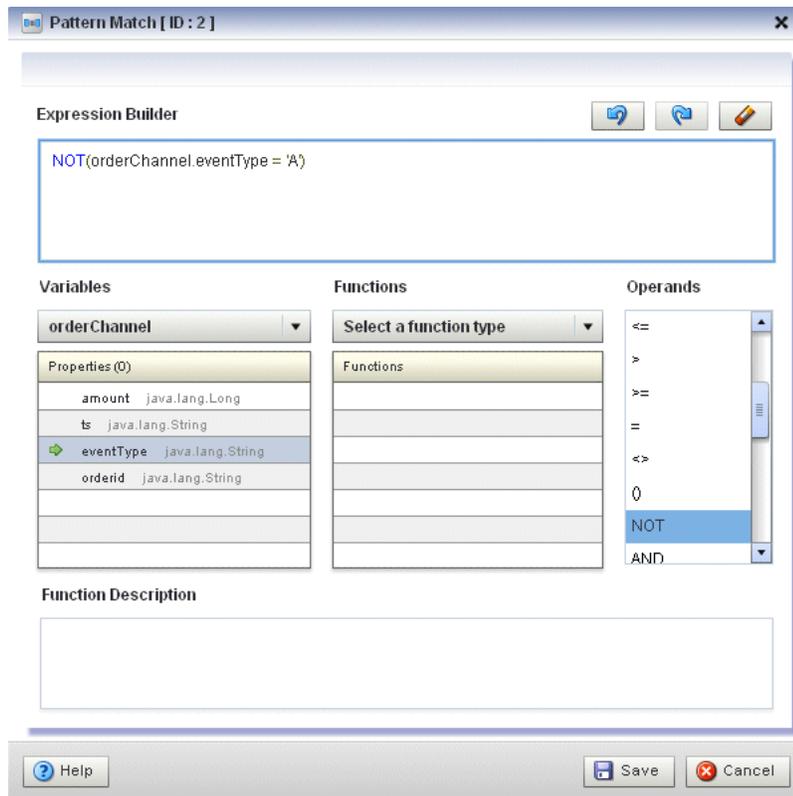
Properties (1)	
CustOrder	orderChannel.eventType = 'C'

Generated Pattern Match Statement

```
SELECT * FROM orderChannel MATCH_RECOGNIZE (PARTITION BY orderid PATTERN (CustOrder
NoApproval*? Shipment) DEFINE CustOrder AS orderChannel.eventType = 'C' AS Orders
```

Help Validate Save Cancel

24. Enter **NoApproval** in the **Object Name** field.
25. Click the Expression Builder button and configure the Expression Builder as follows (see [Figure 3–24](#)):
 - In the **Variables** list, double-click **eventType**.
 - In the **Operands** list, double-click **=**.
 - After the **=** operand, enter the value **'A'**.
 - Place parenthesis around the expression.
 - Place the insertion bar at the beginning of the expression, outside the open parenthesis.
 - In the **Operands** list, double-click **NOT**.

Figure 3–26 Expression Builder: NoApproval

26. Click **Save**.

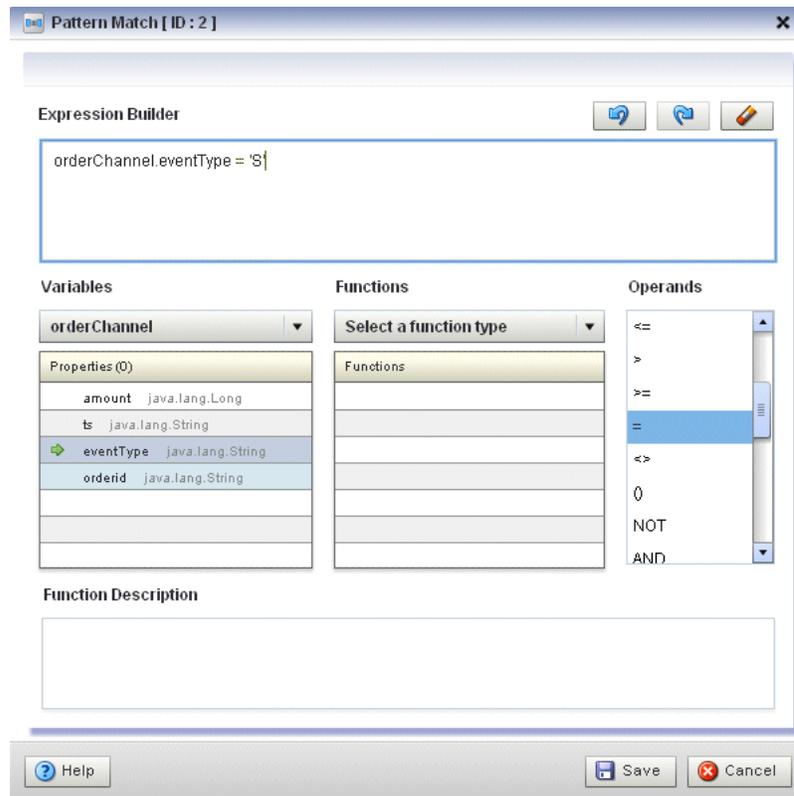
27. Click the Plus Sign button.

The condition definition is added to the Object List.

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

29. Click the Expression Builder button and configure the Expression Builder as follows (see [Figure 3–24](#)):

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

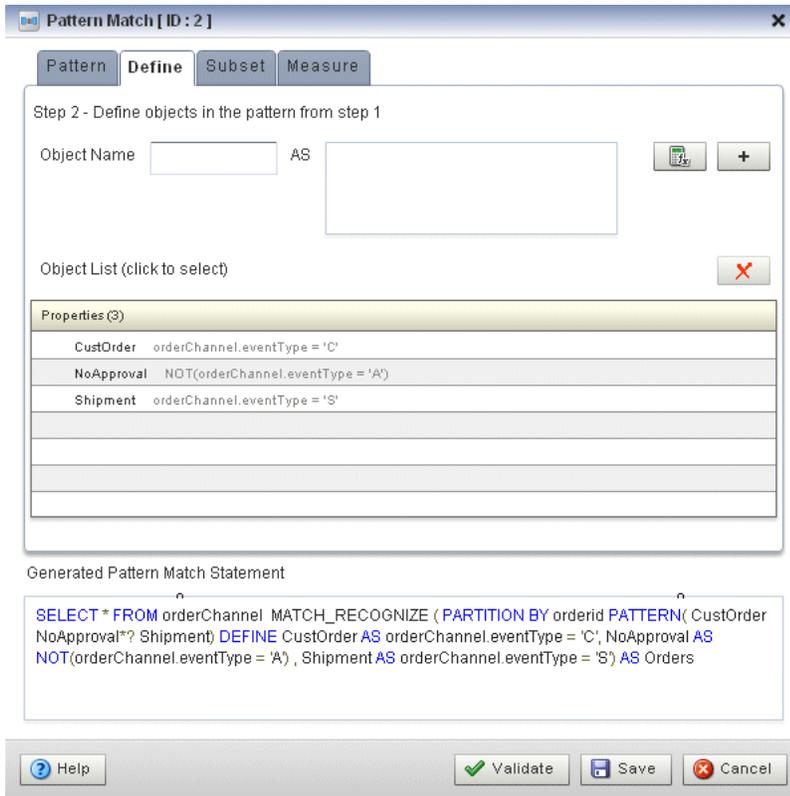
Figure 3–27 Expression Builder: Shipment

30. Click **Save**.

31. Click the Plus Sign button.

The Define tab appears as [Figure 3–28](#) shows.

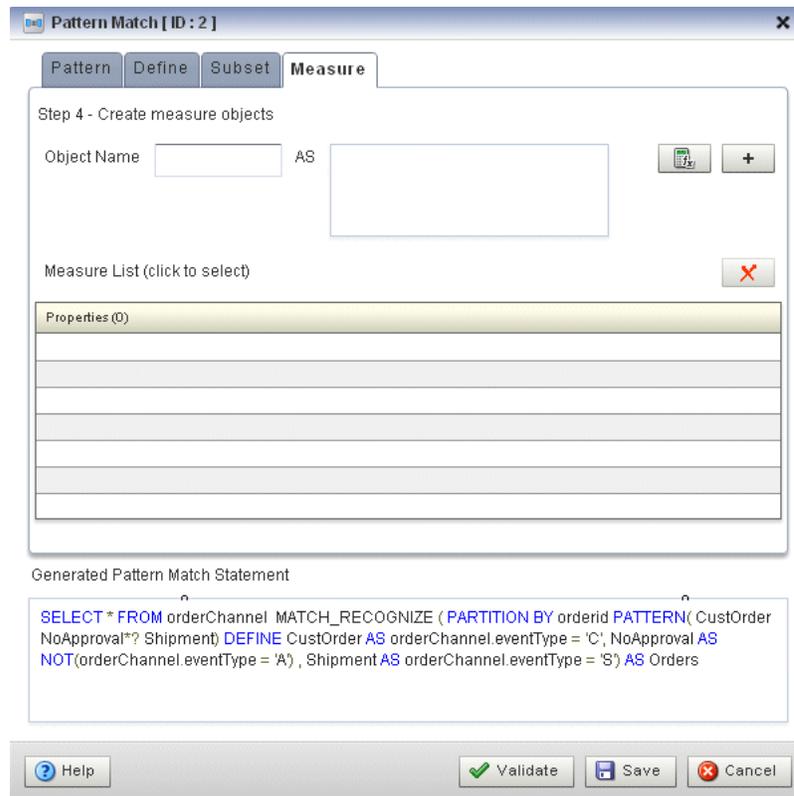
Figure 3–28 Pattern Configuration Dialog: Define Tab Complete



32. Click the **Measure** tab.

The Measure tab appears as [Figure 3–29](#) shows.

Figure 3–29 Measure Tab



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

Use the Measure tab to specify the following:

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

For more information, see:

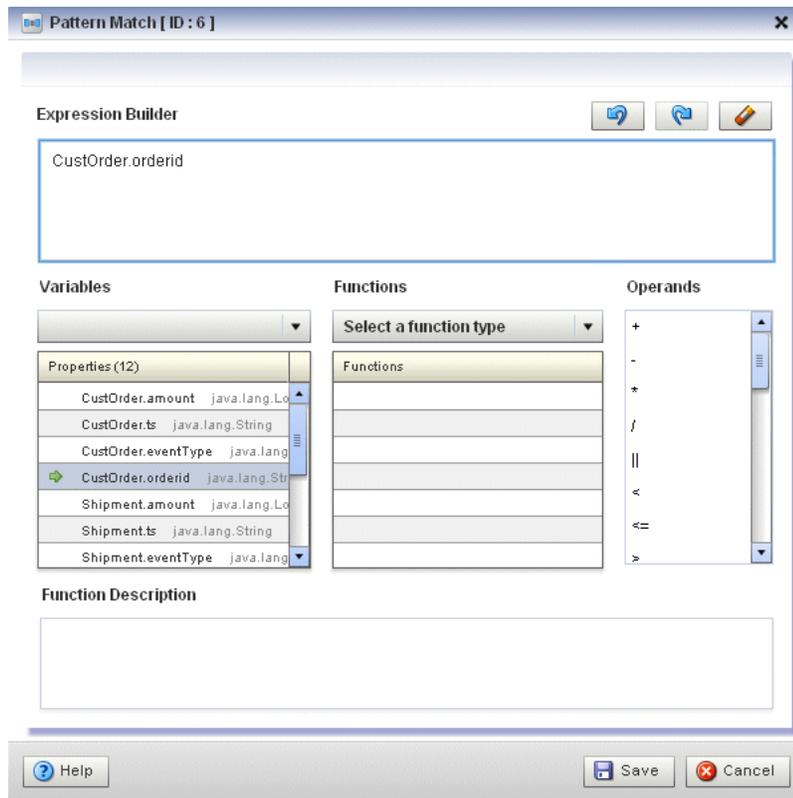
- "MEASURES Clause" in the *Oracle CEP CQL Language Reference*
- "MATCH_RECOGNIZE Condition" in the *Oracle CEP CQL Language Reference*

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

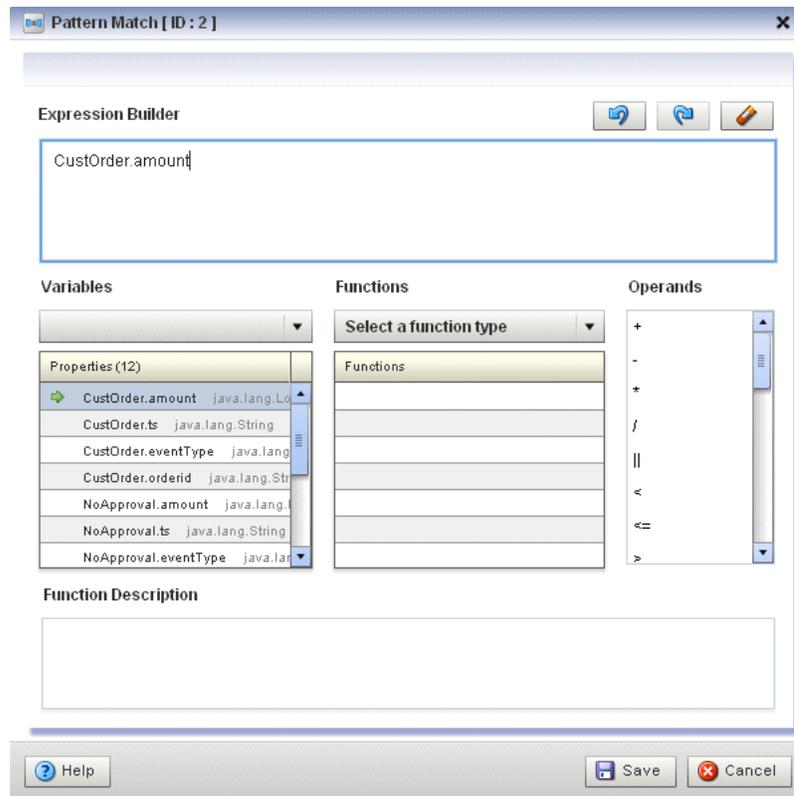
34. Click the Expression Builder button and configure the Expression Builder as follows (see [Figure 3–24](#)):

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

Figure 3–30 Expression Builder:orderid



35. Click **Save**.
36. Click the Plus Sign button.
37. Enter **amount** in the **Object Name** field.
38. Click the Expression Builder button and configure the Expression Builder as follows (see [Figure 3–24](#)):
 - In the **Variables** list, double-click **CustOrder.amount**.

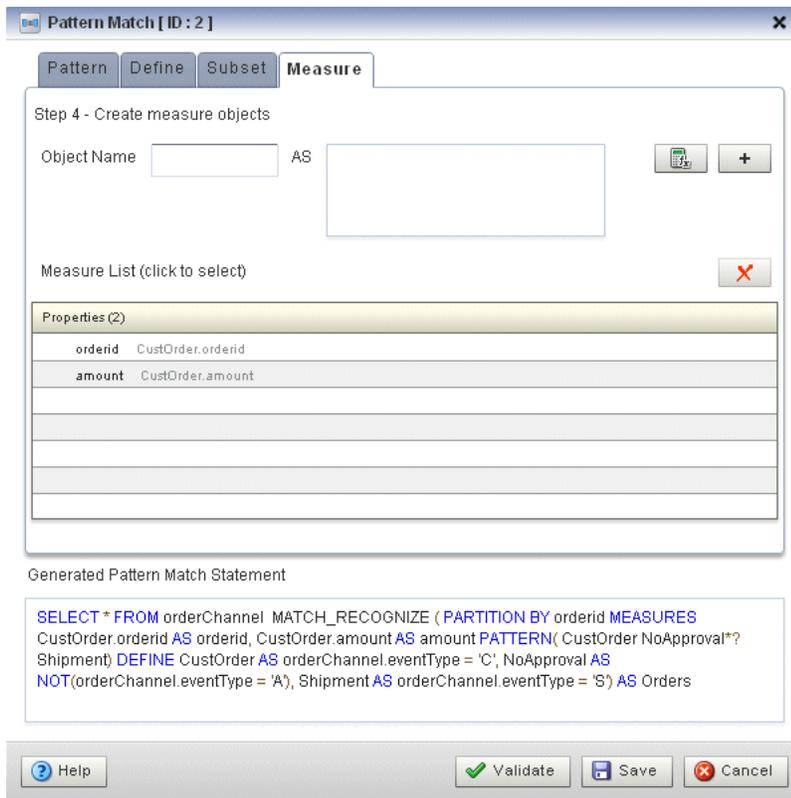
Figure 3–31 Expression Builder: amount

39. Click **Save**.

40. Click the Plus Sign button.

The Measure tab appears as [Figure 3–32](#) shows.

Figure 3–32 Measure Tab: Complete



41. Click **Save**.

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

The Select configuration screen appears as [Figure 3–33](#) shows.

Figure 3–33 Select Configuration Dialog: Project Tab

Step 1 - Project

Distinct Results Target Event Type **Select or Input Event Type**

Source Properties (select from here)

Select a source

Properties (0)

Source Properties

Select List (0)

Project Expression

Generated CQL Statement

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

Help Validate Save Cancel

43. Configure the Project tab as follows:

- Select **AlertEvent** from the **Select or Input Event Type** pull-down menu.
- Select **Orders** from the **Select a source** pull-down menu.

44. Double-click **orderid** in the **Properties** list and select **orderid** from the **Select or Input Alias** pull-down menu.

45. Click the Plus Sign button to add the property to the Generated CQL Statement.

46. Double-click **amount** in the **Properties** list and select **amount** from the **Select or Input Alias** pull-down menu.

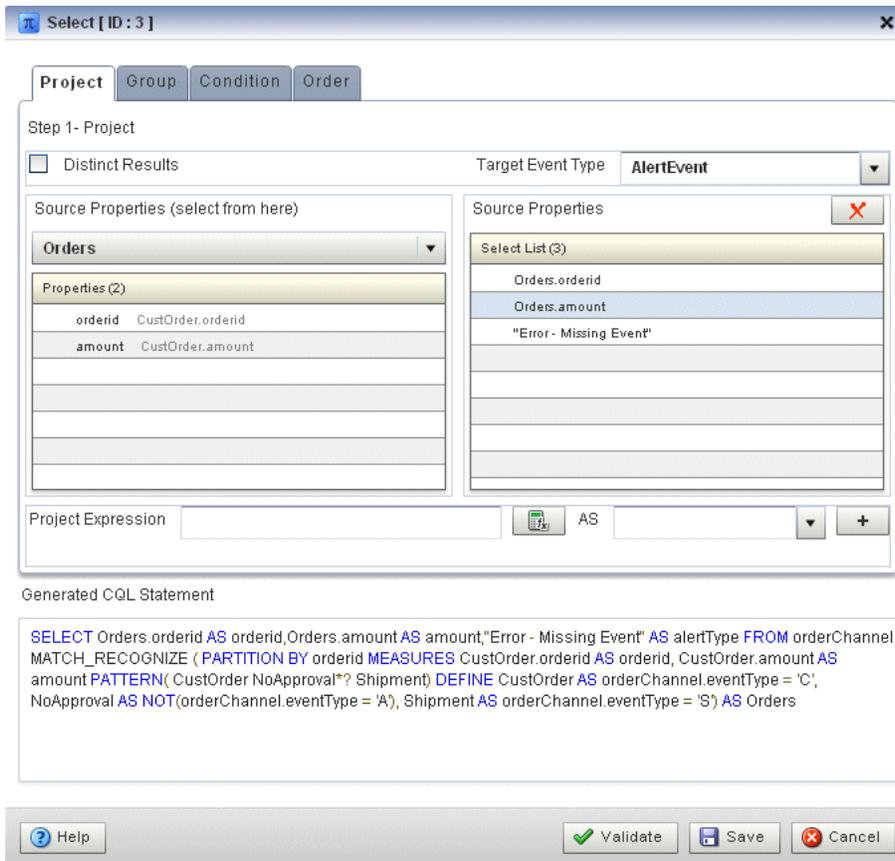
47. Click the Plus Sign button to add the property to the Generated CQL Statement.

48. Click in the Project Expression field and enter the value "Error - Missing Approval" and select **alertType** from the **Select or Input Alias** pull-down menu.

49. Click the Plus Sign button to add the property to the Generated CQL Statement.

The Project tab appears as [Figure 3–34](#) shows.

Figure 3–34 Select Configuration Dialog: Project Tab Complete



50. Click **Save**.
51. Click **Save Query**.
52. Double-click the **Output** icon.

The Output configuration screen appears as [Figure 3–83](#) shows.

Figure 3–35 Output Configuration Dialog

Type Query

Query Name

Enable true false

View

View Name

View Schema

Project List

Properties (3)	
1	Orders.orderid:orderid
2	Orders.amount:amount
3	"Error - Missing Event".alertType

Generated CQL Statement

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

Help Inject Rule Replace Rule Validate Save Cancel

53. Configure the Output as follows:

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

54. Click **Inject Rule**.

The Inject Rule Confirmation dialog appears as [Figure 3–57](#) shows.

Figure 3–36 Inject Rule Confirmation Dialog

Confirmation

Rule has been successfully added.

OK

55. Click **OK**.

The Query Wizard adds the rule to the cqlProc processor.

56. Click **Save**.

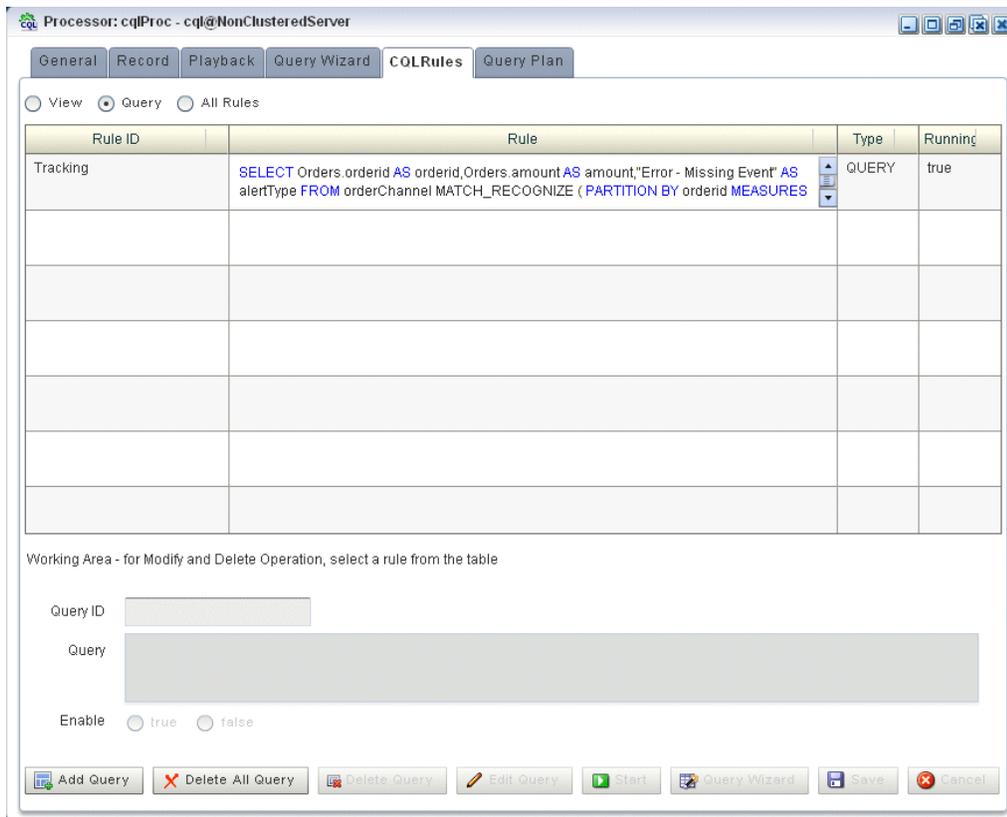
57. Click on the **CQL Rules** tab.

The CQL Rules tab appears as [Figure 3–58](#) shows.

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

Confirm that your Tracking query is present.

Figure 3–37 CQL Rules Tab With Tracking Query

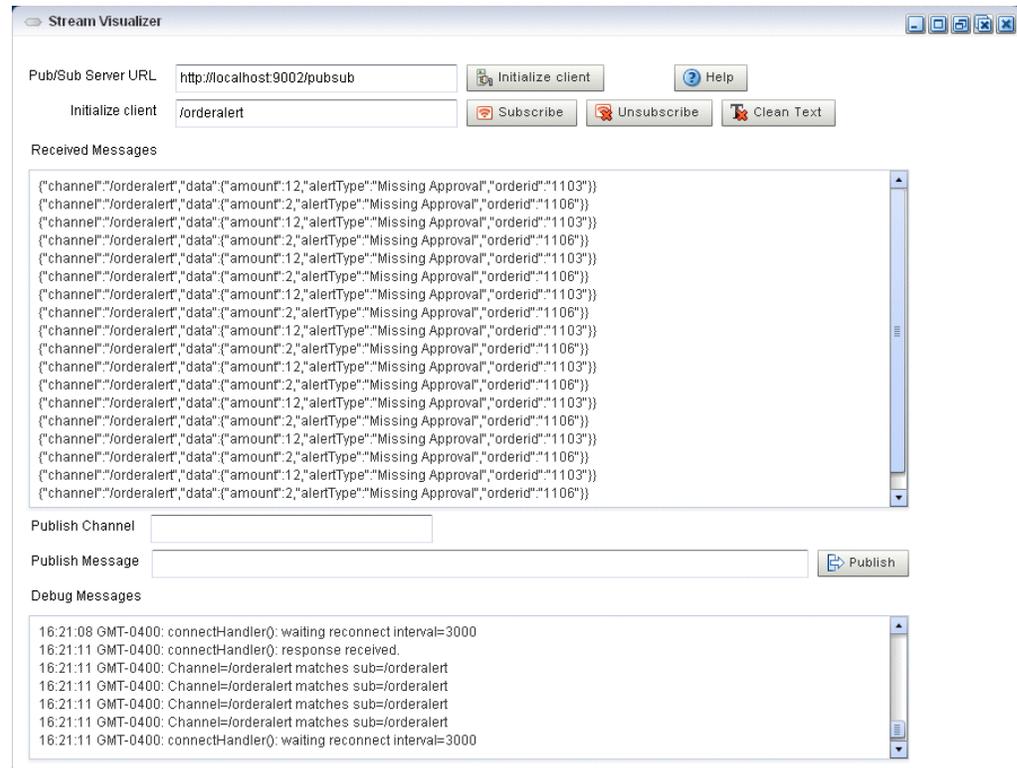


To test the missing event query:

1. To simulate the data feed for the missing event query, open a new command window and set your environment as described in [Section 3.5, "Setting Your Development Environment."](#)
2. 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`.
3. Run the load generator using the `data-aggre.prop` properties file:
 - a. On Windows:


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


```
prompt> runloadgen.sh data-aggre.prop
```
4. In the Oracle CEP Visualizer, click the **ViewStream** button in the top pane. The Stream Visualizer screen appears as [Figure 3–38](#) shows.

Figure 3–38 Stream Visualizer: Showing Missing Events

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

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

3.10.4.2 Creating the Moving Average Query

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

You do this in two steps:

- First, you create a view (the Oracle CQL equivalent of a subquery) that serves as the source of the moving average query.
 - See ["To create a view source for the moving average query:"](#) on page 3-75.
- Second, you create the moving average query using the source view.
 - See ["To create the moving average query using the view source:"](#) on page 3-93.
- Finally, you test the moving average query.
 - See ["To test the moving average query:"](#) on page 3-118.

To create a view source for the moving average query:

1. If the CQL Oracle CEP instance is not already running, follow the procedure in [Section 3.10.1, "Running the CQL Example"](#) to start the server.

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

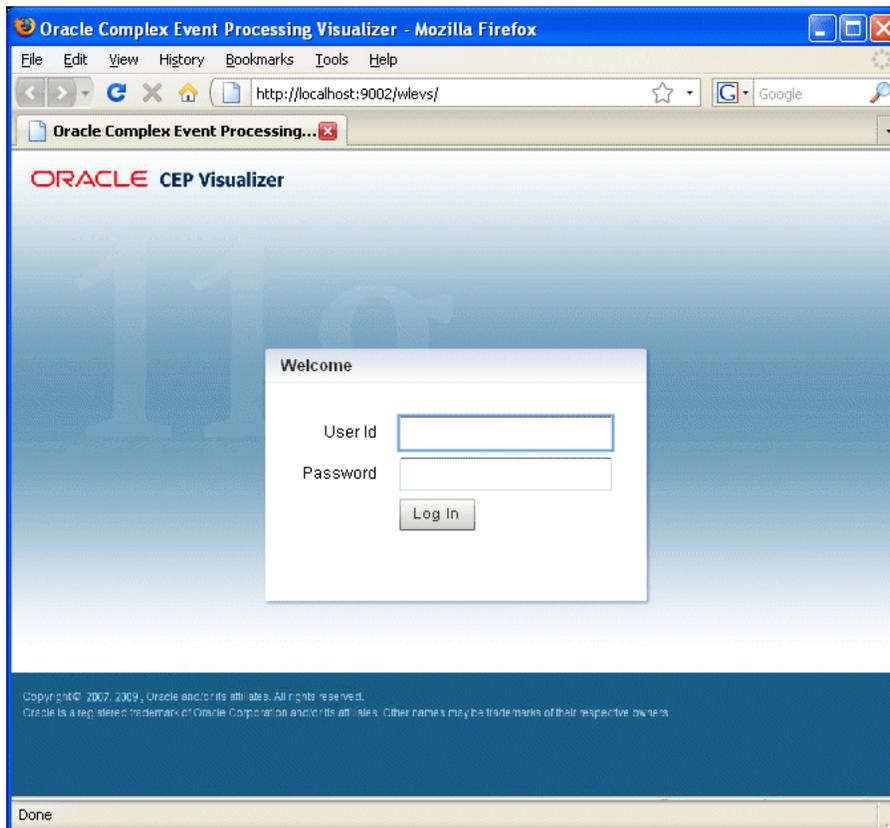
2. Invoke the following URL in your browser:

`http://host:port/wlevs`

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

The Logon screen appears as [Figure 3–39](#) shows.

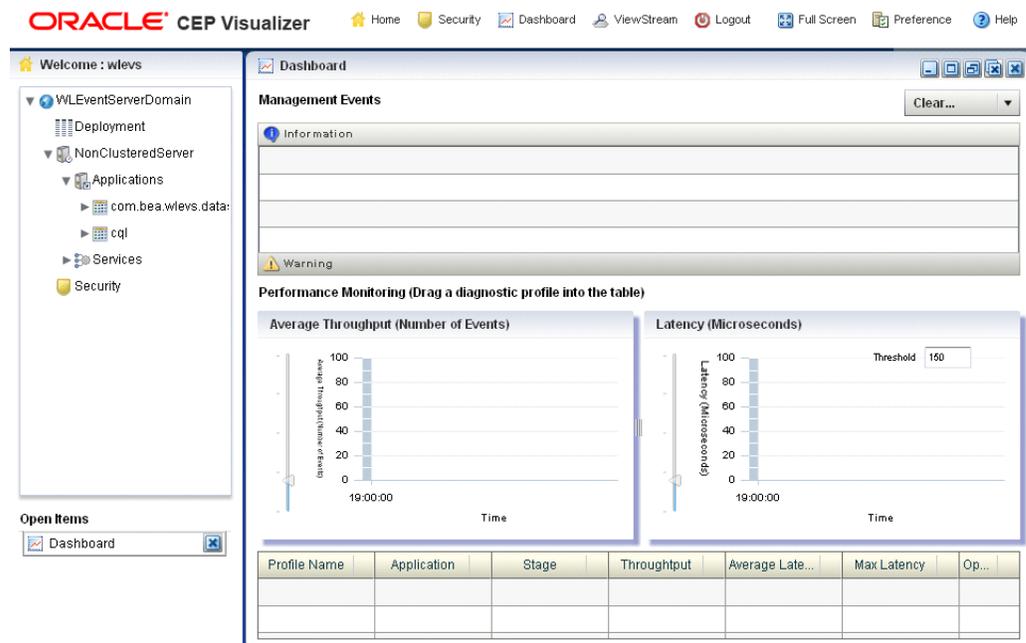
Figure 3–39 Oracle CEP Visualizer Logon Screen



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

The Oracle CEP Visualizer dashboard appears as [Figure 3–40](#) shows.

Figure 3–40 Oracle CEP Visualizer Dashboard

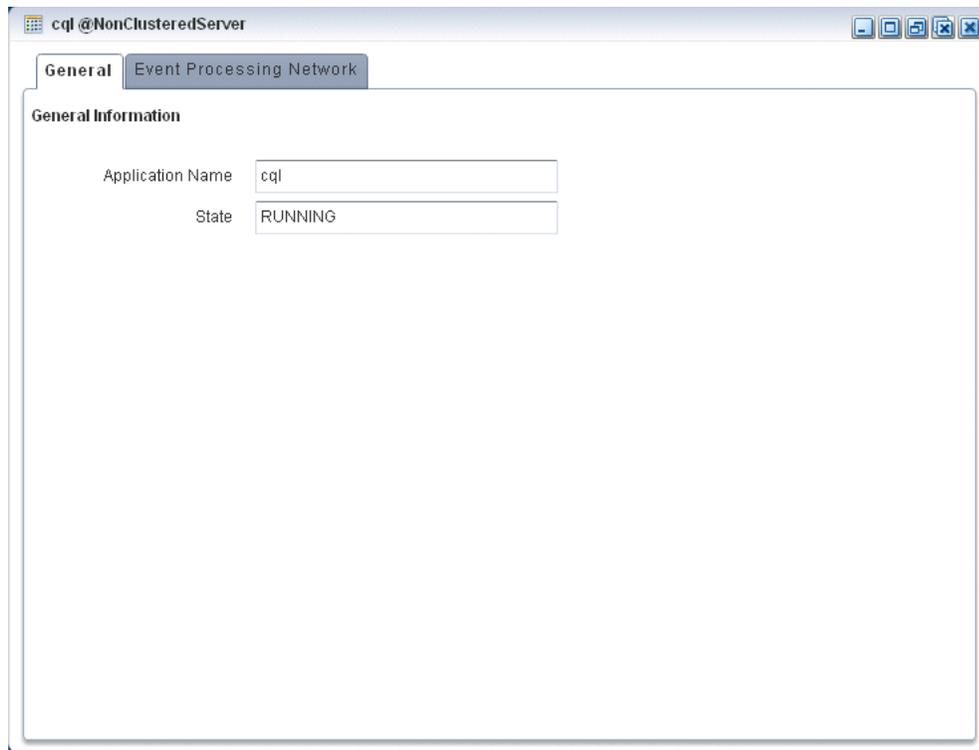


For more information about the Oracle CEP Visualizer user interface, see "Understanding the Oracle CEP Visualizer User Interface" in the *Oracle CEP Visualizer User's Guide*.

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

The CQL application screen appears as [Figure 3–41](#) shows.

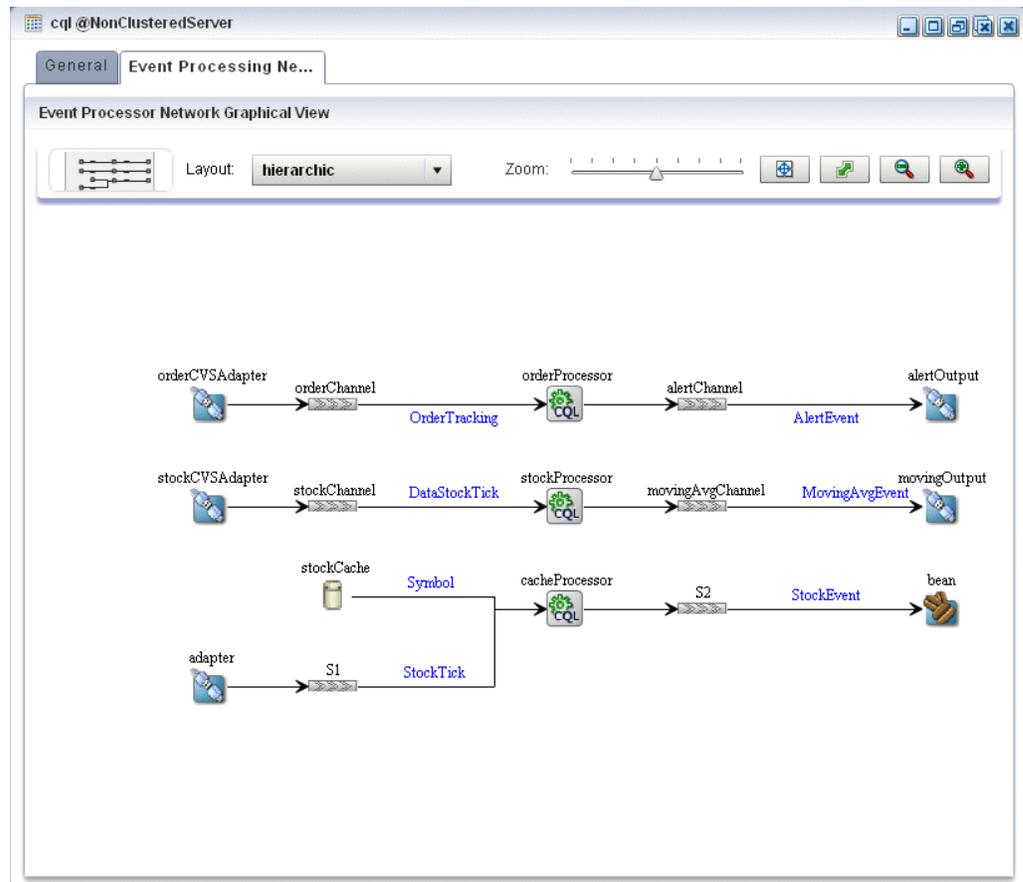
Figure 3–41 CQL Application Screen: General Tab



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

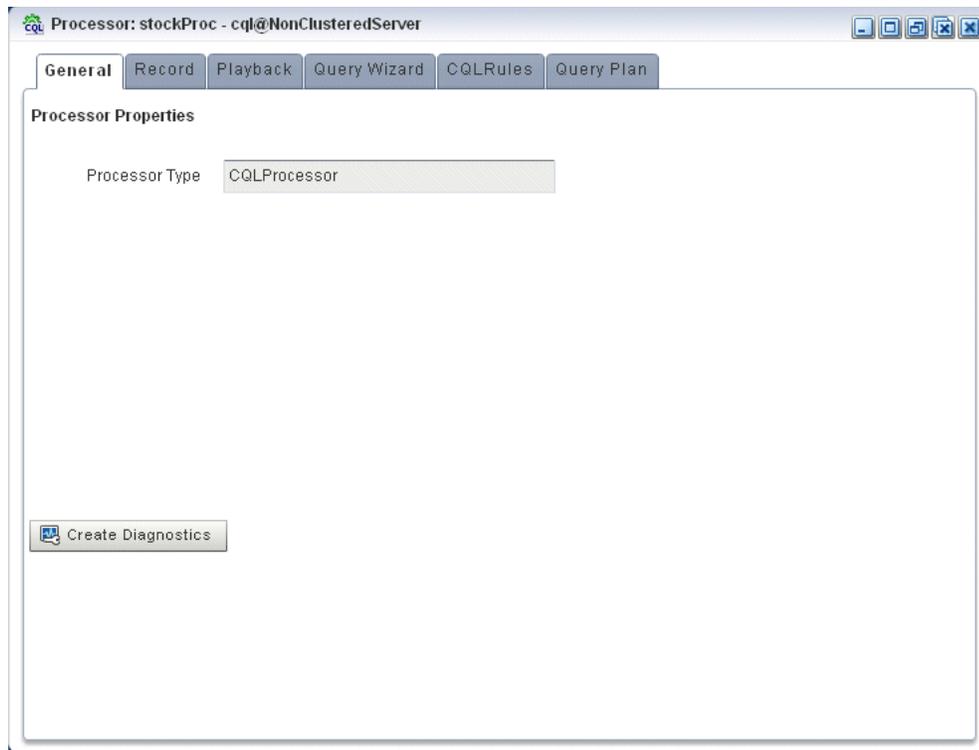
The Event Processing Network screen appears as [Figure 3–42](#) shows.

Figure 3–42 CQL Application: Event Processing Network Tab

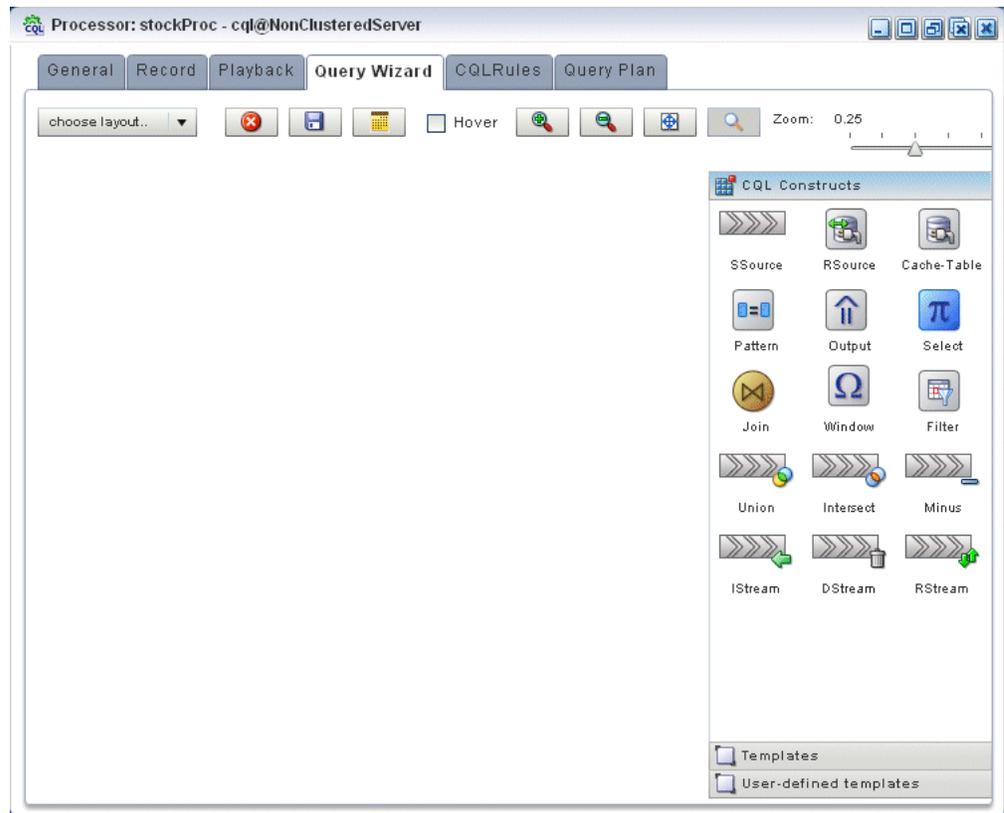


7. Double-click the **stockProcessor** Oracle CQL processor icon.
The Oracle CQL processor screen appears as [Figure 3–43](#) shows.

Figure 3–43 Oracle CQL Processor: General Tab



8. Select the **Query Wizard** tab.
The Query Wizard screen appears as [Figure 3–44](#) shows.

Figure 3–44 Oracle CQL Processor: Query Wizard Tab

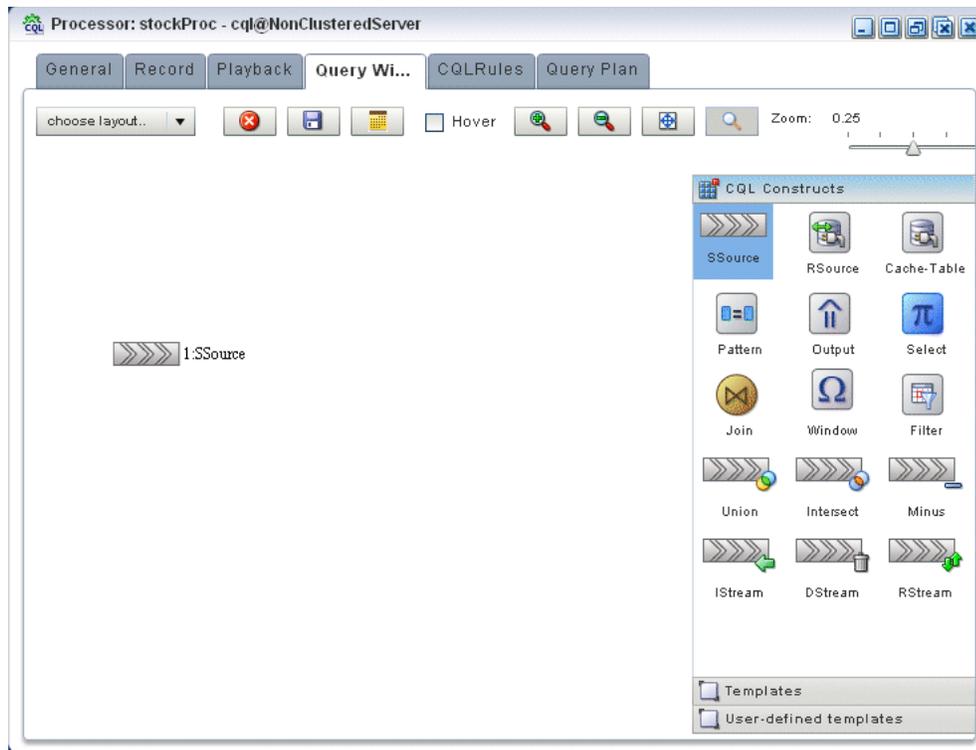
You can use the Oracle CQL Query Wizard to construct an Oracle CQL query from a template or from individual Oracle CQL constructs.

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

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

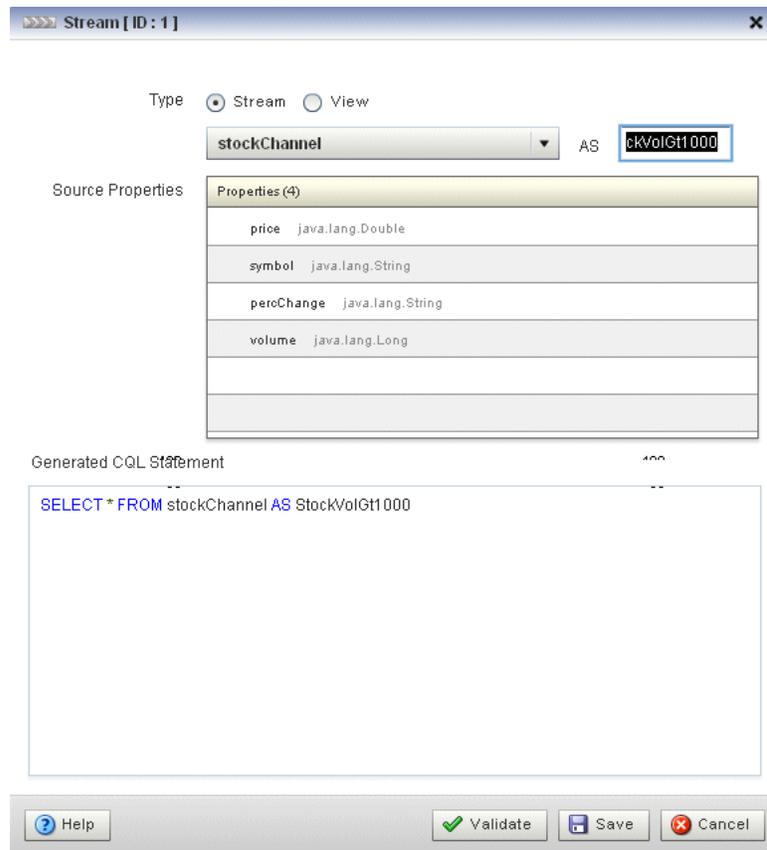
9. Click and drag an SSource icon (Stream Source) from the CQL Constructs palette and drop it anywhere in the Query Wizard canvas as [Figure 3–65](#) shows.

Figure 3–45 Query Wizard: SSource



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

The SSource configuration screen appears as [Figure 3–46](#) shows.

Figure 3–46 SSource Configuration Dialog

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

11. Configure the SSource as follows:

- Select **Stream** as the Type.
 - The source of our view is the `stockChannel` stream.
- Select **stockChannel** from the **Select a source** pull-down menu.
- Enter the alias `StockVolGt1000` in the **AS** field.

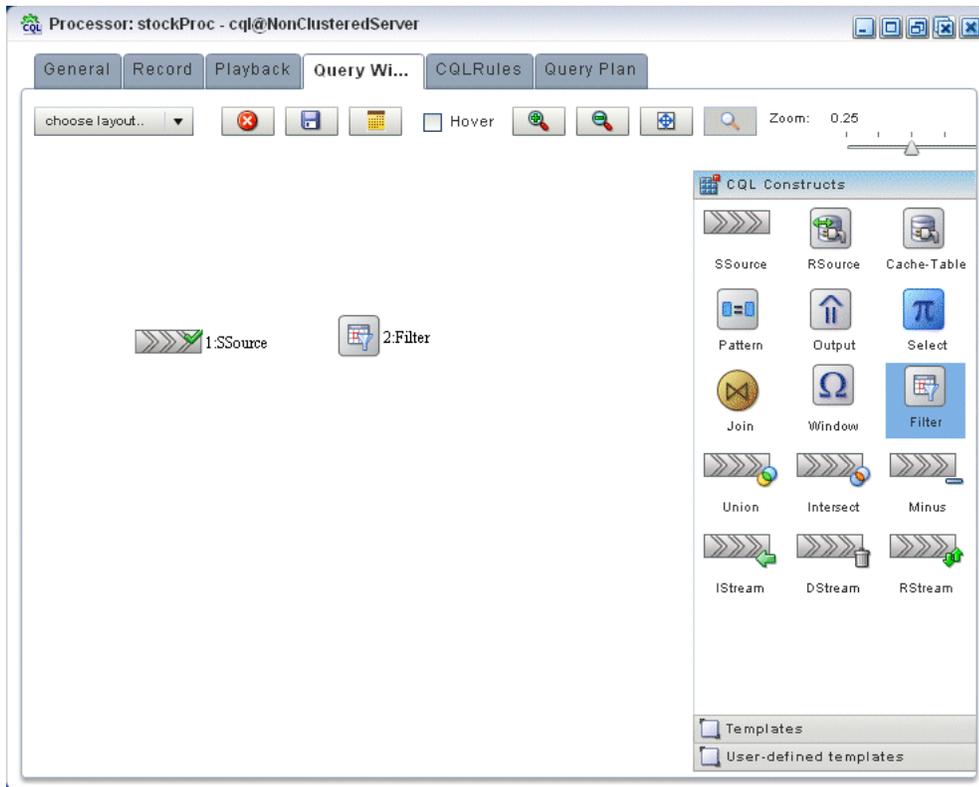
12. Click Save.

13. Click Save Query.

Next, we will add an Oracle CQL filter.

14. Click and drag a Filter icon from the CQL Constructs palette and drop it anywhere in the Query Wizard canvas as [Figure 3–47](#) shows.

Figure 3–47 Query Wizard: Filter



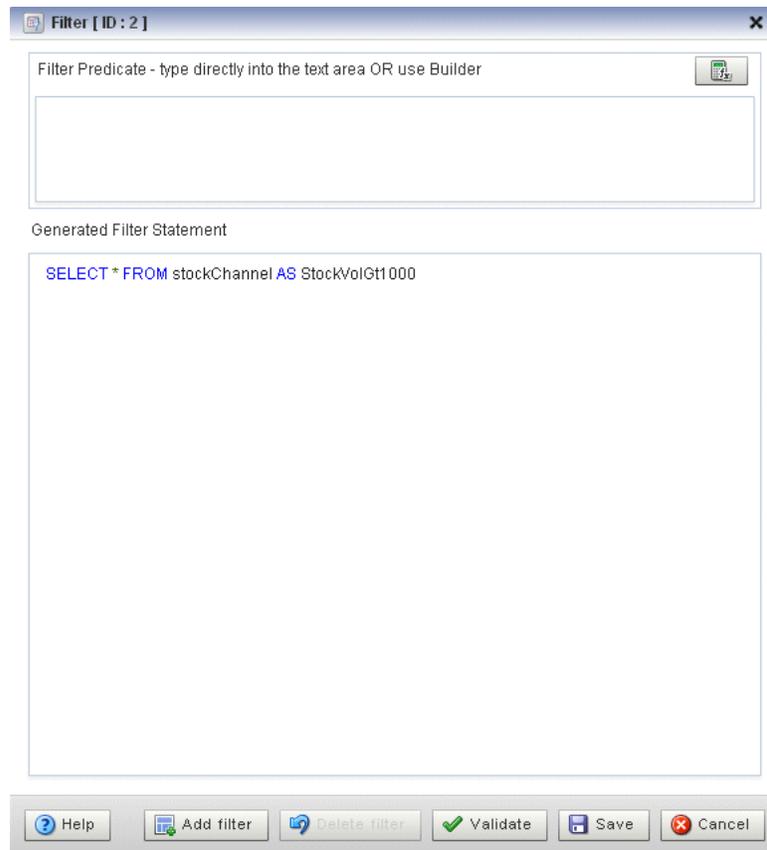
15. Click on the SSource icon and drag to the Window icon to connect the Oracle CQL constructs as [Figure 3–48](#) shows.

Figure 3–48 Connecting the SSource and Filter Icons



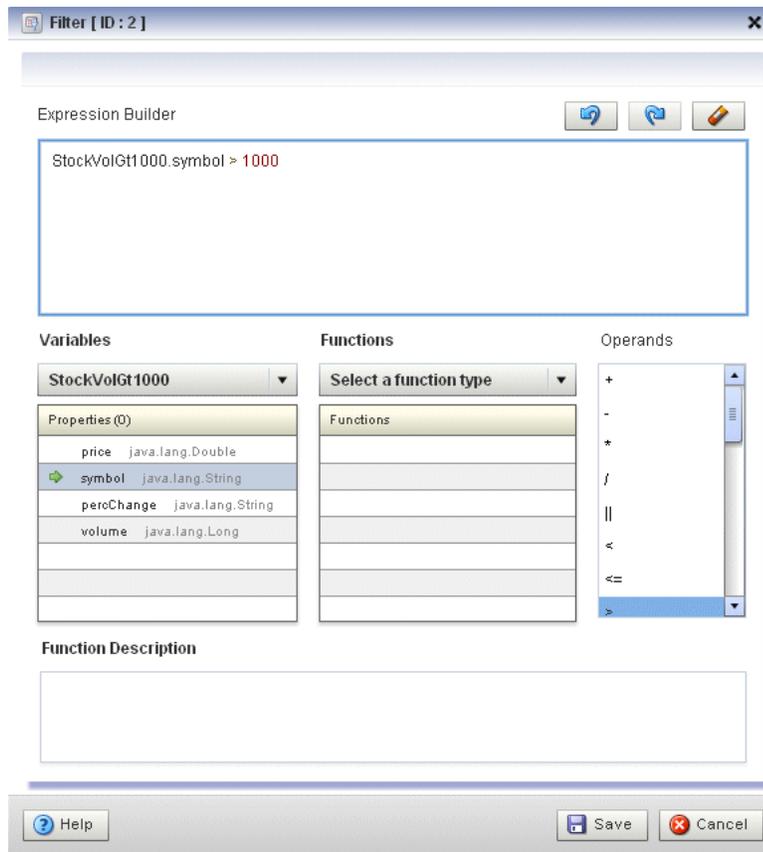
16. Double-click the **Filter** icon.

The Filter configuration screen appears as [Figure 3–49](#) shows.

Figure 3–49 Filter Configuration Dialog

17. Click the Expression Builder button.

The Expression Builder dialog appears as [Figure 3–50](#) shows.

Figure 3–50 Filter Expression Builder

18. Configure the Expression Builder as follows:

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

19. Click **Save**.

20. Click **Add Filter**.

The Query Wizard adds the expression to the Generated CQL Statement as [Figure 3–51](#) shows.

Figure 3–51 Filter Configuration Dialog: After Adding the Filter

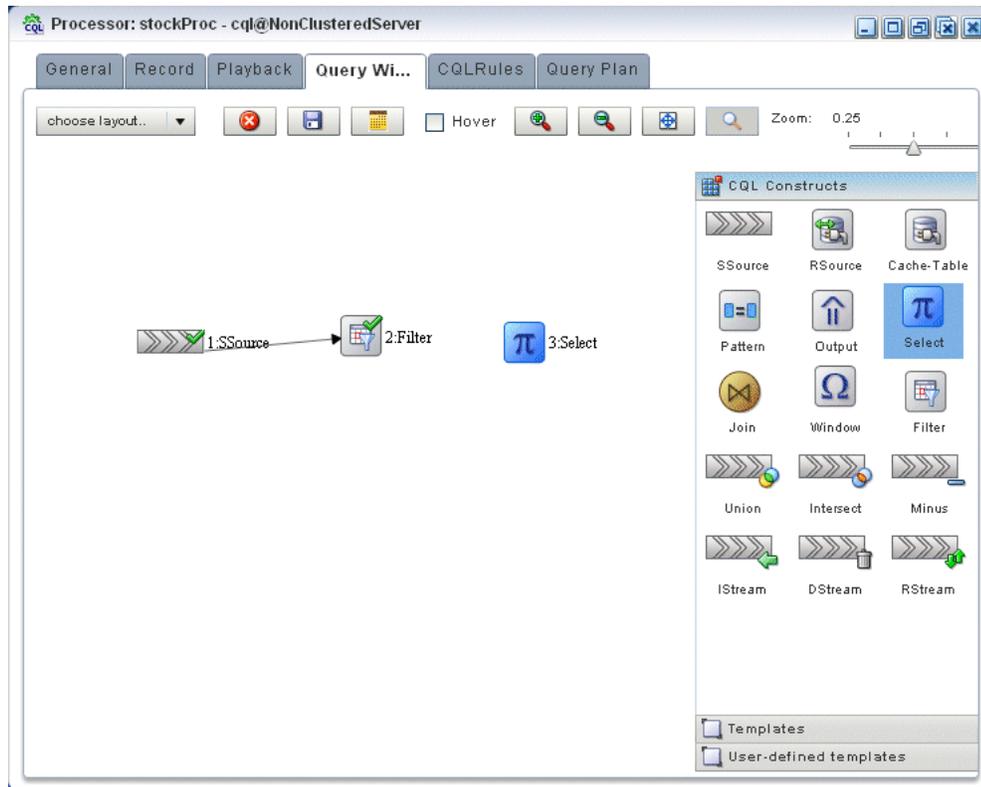
21. Click **Save**.

22. Click **Save Query**.

Next we want to add a select statement.

23. Click and drag a Select icon from the CQL Constructs palette and drop it anywhere in the Query Wizard canvas as [Figure 3–52](#) shows.

Figure 3–52 Query Wizard: Select



24. Click on the **Filter** icon and drag to the **Select** icon to connect the Oracle CQL constructs.

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

The Select configuration screen appears as [Figure 3–53](#) shows.

Figure 3–53 Select Configuration Dialog

Step 1- Project

Distinct Results Target Event Type **Select or Input Event Type**

Source Properties (select from here)

StockVolGt1000

Properties (4)

price	java.lang.Double
symbol	java.lang.String
percChange	java.lang.String
volume	java.lang.Long

Source Properties

Select List (0)

Project Expression AS

Generated CQL Statement

```
SELECT * FROM stockChannel AS StockVolGt1000 WHERE StockVolGt1000.symbol > 1000
```

Help Validate Save Cancel

We want to select `price`, `symbol`, and `volume` from our `StockVolGt1000` stream.

26. Configure the Select as follows:

- Select **StockVolGt1000** from the **Select a source** pull-down menu.
 - Select the **price** property and click the Plus Sign button.
- The Query Wizard adds the property to Generated CQL Statement
- Repeat for the **symbol** and **volume** properties.

The Select configuration dialog appears as [Figure 3–54](#) shows.

Figure 3–54 Select Configuration Dialog: Properties Selected

Select [ID : 3]

Project Group Condition Order

Step 1- Project

Distinct Results Target Event Type Select or Input Event Type

Source Properties (select from here)

StockVolGt1000

Properties (4)

price	java.lang.Double
symbol	java.lang.String
percChange	java.lang.String
volume	java.lang.Long

Source Properties

Select List (3)

StockVolGt1000.price
StockVolGt1000.symbol
StockVolGt1000.volume

Project Expression AS

Generated CQL Statement

```
SELECT StockVolGt1000.price,StockVolGt1000.symbol,StockVolGt1000.volume FROM stockChannel AS
StockVolGt1000 WHERE StockVolGt1000.symbol > 1000
```

Help Validate Save Cancel

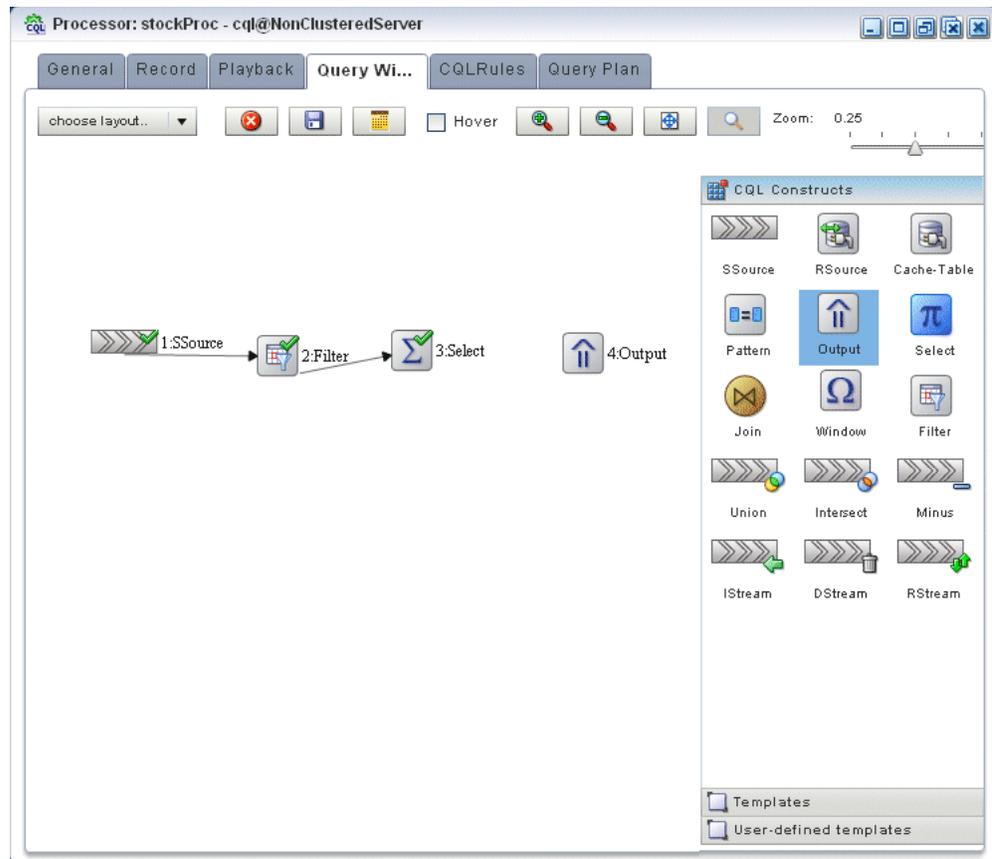
27. Click **Save**.

28. Click **Save Query**.

Finally, we will add an Output.

29. Click and drag an Output icon from the CQL Constructs palette and drop it anywhere in the Query Wizard canvas as [Figure 3–55](#) shows.

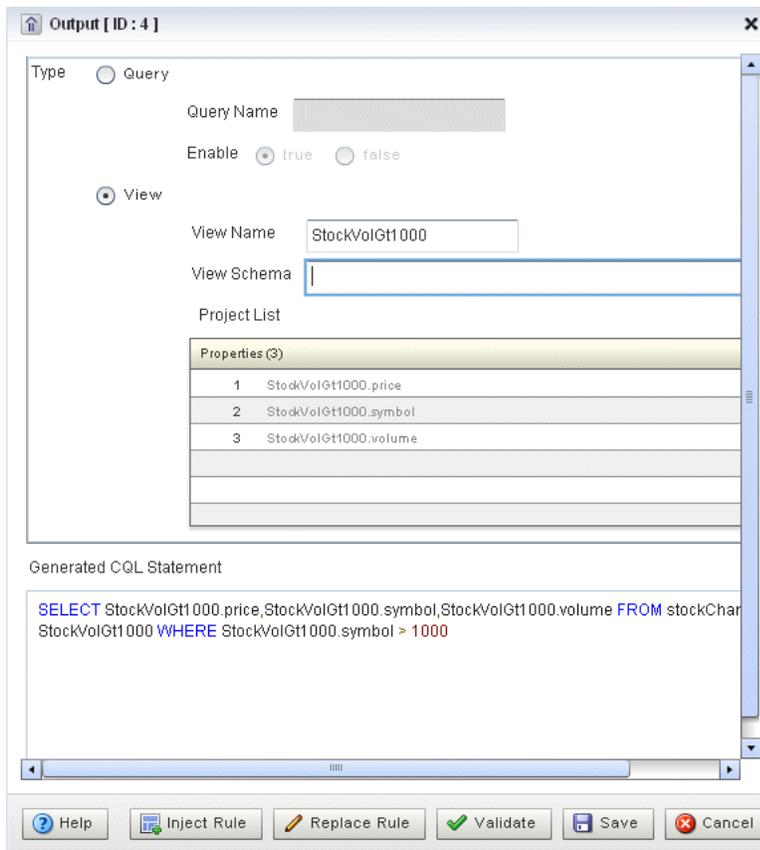
Figure 3–55 Query Wizard: Output



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

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

The Output configuration screen appears as [Figure 3–56](#) shows.

Figure 3–56 Output Configuration Dialog

32. Configure the Output as follows:

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

We will let the Oracle CEP server define the view schema for us.

33. Click **Inject Rule**.

The Inject Rule Confirmation dialog appears as [Figure 3–57](#) shows.

Figure 3–57 Inject Rule Confirmation Dialog

34. Click **OK**.

The Query Wizard adds the rule to the `cqlProc` processor.

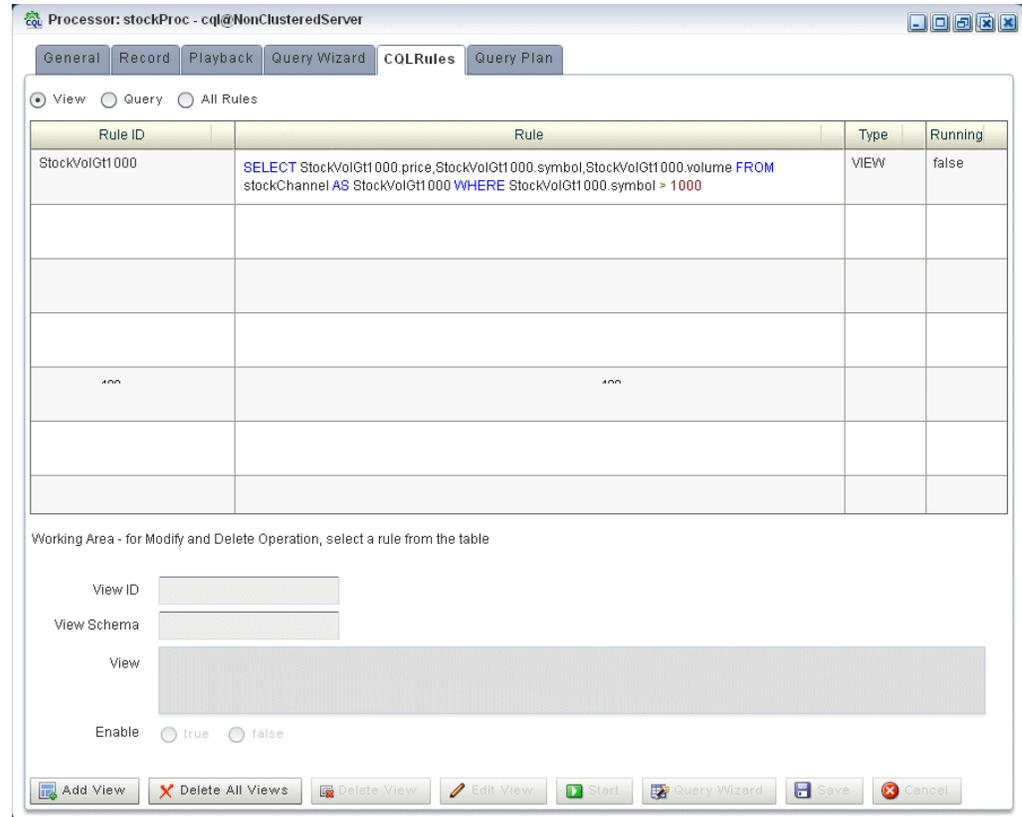
35. Click **Save**.

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

The CQL Rules tab appears as [Figure 3–58](#) shows.

37. Click on the View radio button.

Confirm that your `StockVolGt1000` view is present.

Figure 3–58 CQL Rules Tab With View StockVolGt1000**To create the moving average query using the view source:**

1. If the CQL Oracle CEP instance is not already running, follow the procedure in [Section 3.10.1, "Running the CQL Example"](#) to start the server.

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

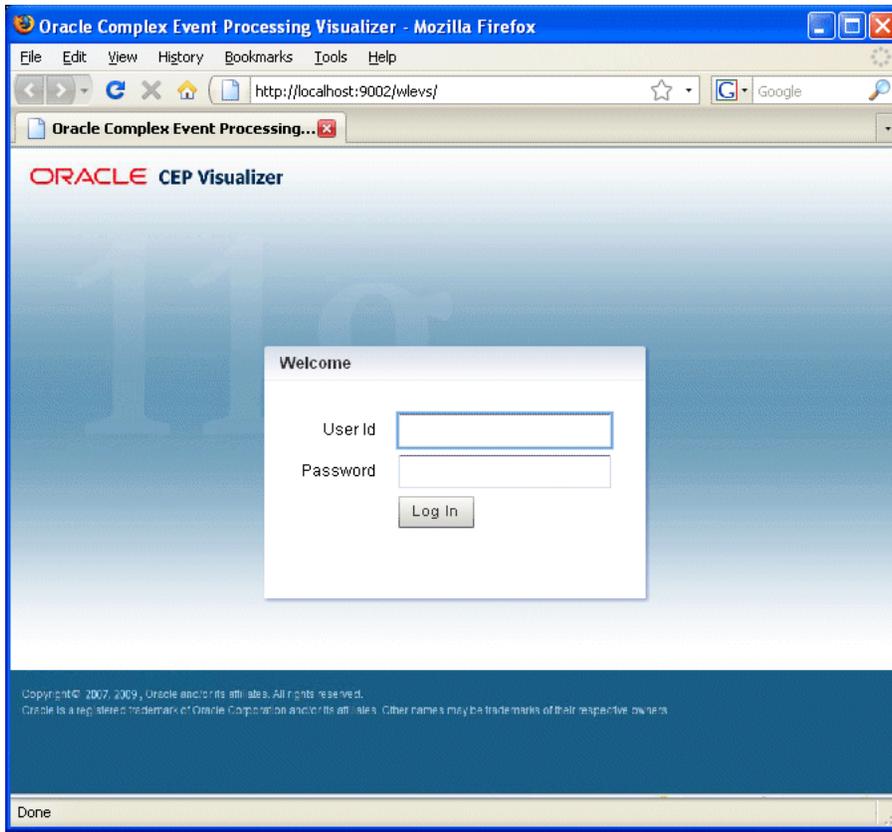
2. Invoke the following URL in your browser:

`http://host:port/wlevs`

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

The Logon screen appears as [Figure 3–59](#) shows.

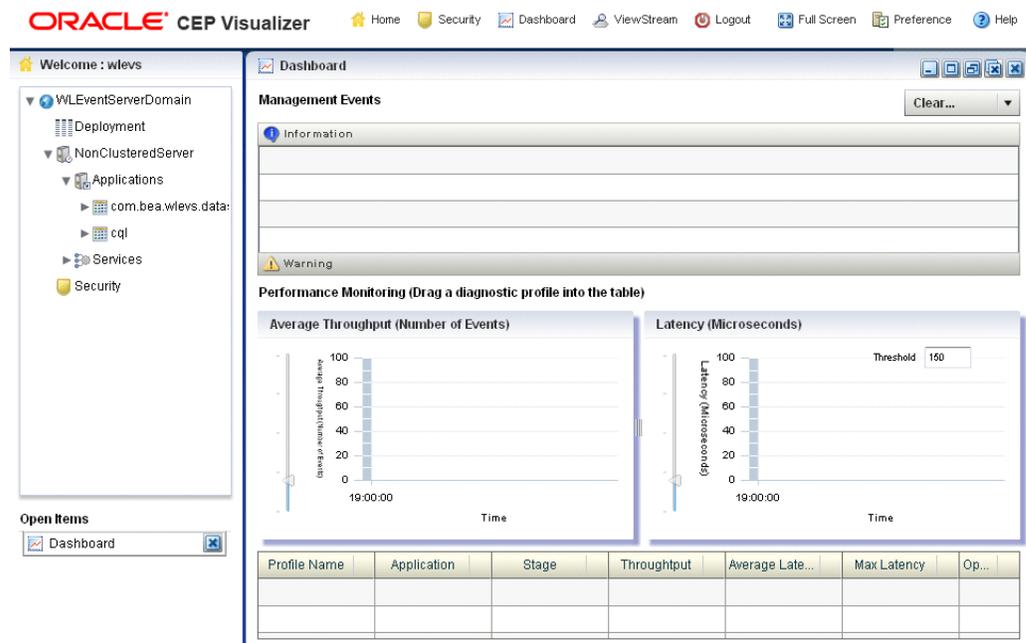
Figure 3–59 Oracle CEP Visualizer Logon Screen



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

The Oracle CEP Visualizer dashboard appears as [Figure 3–40](#) shows.

Figure 3–60 Oracle CEP Visualizer Dashboard

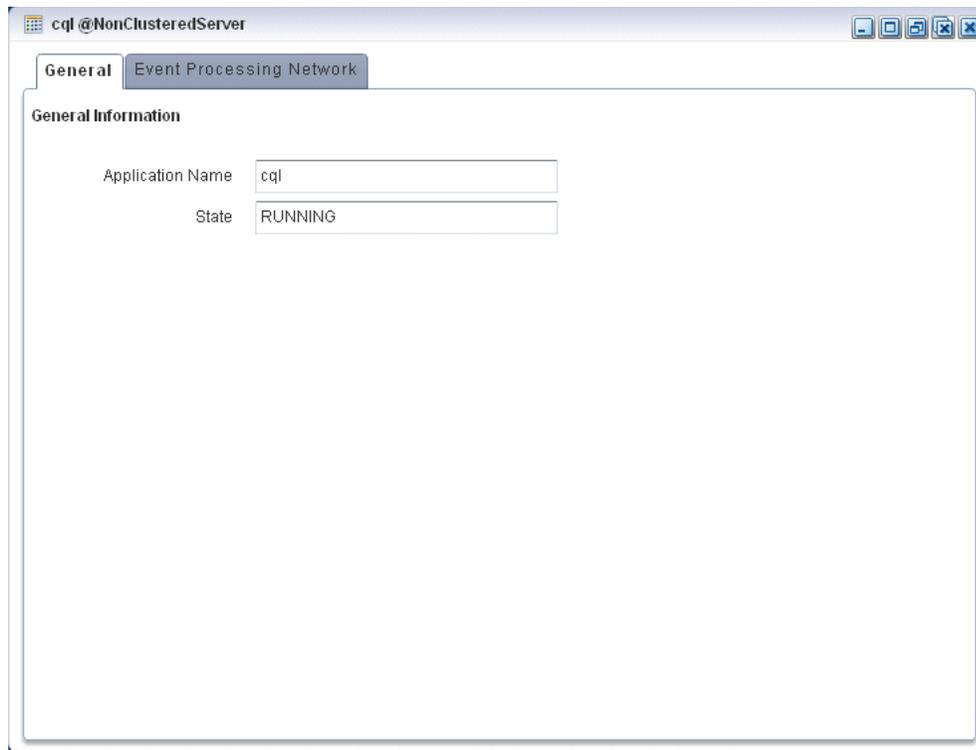


For more information about the Oracle CEP Visualizer user interface, see "Understanding the Oracle CEP Visualizer User Interface" in the *Oracle CEP Visualizer User's Guide*.

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

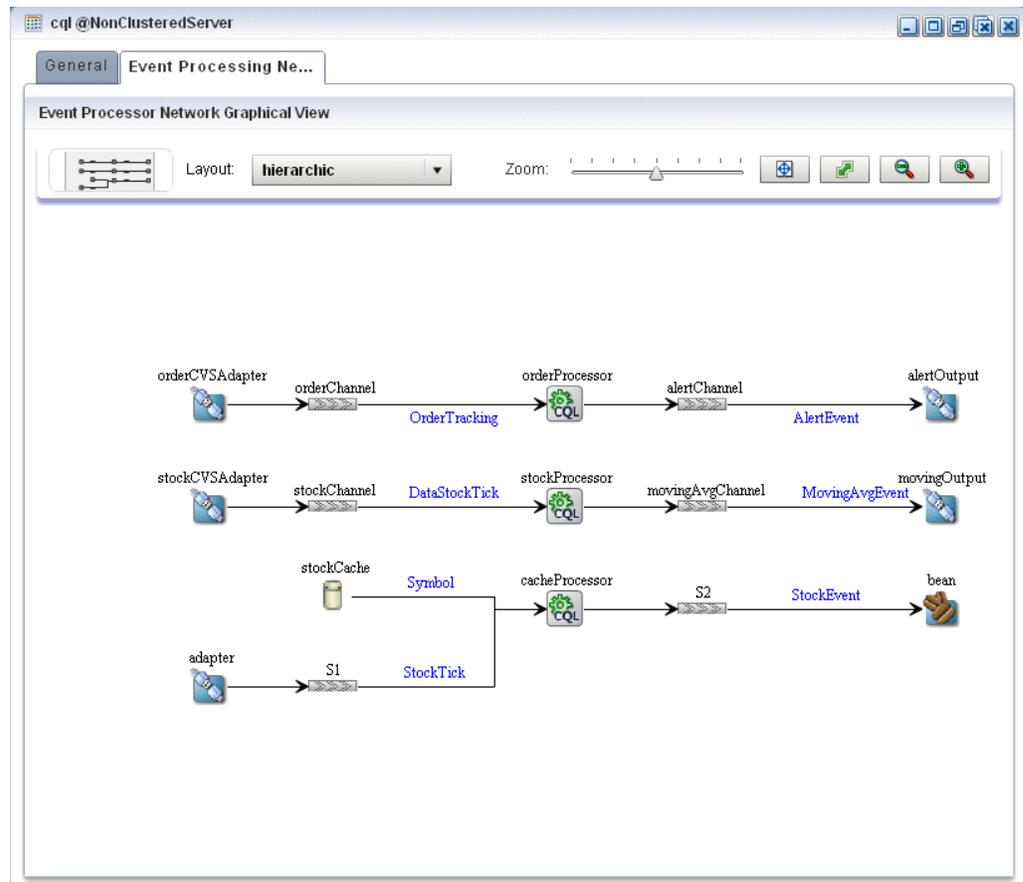
The CQL application screen appears as [Figure 3–41](#) shows.

Figure 3–61 CQL Application Screen: General Tab



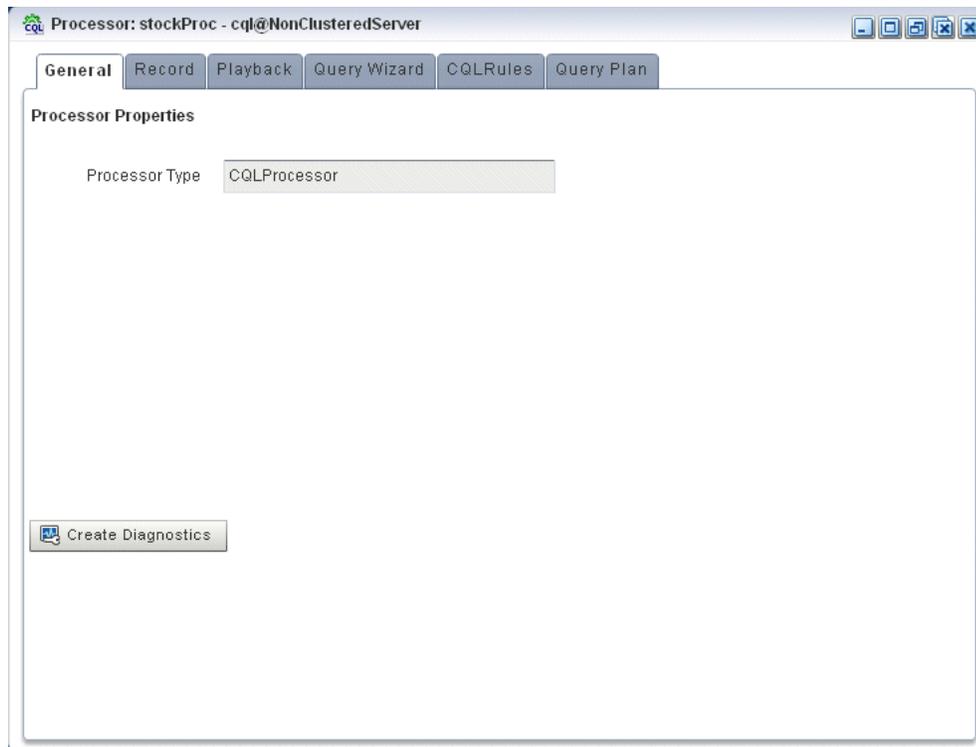
6. Select the **Event Processing Network** tab.

The Event Processing Network screen appears as [Figure 3–42](#) shows.

Figure 3–62 CQL Application: Event Processing Network Tab

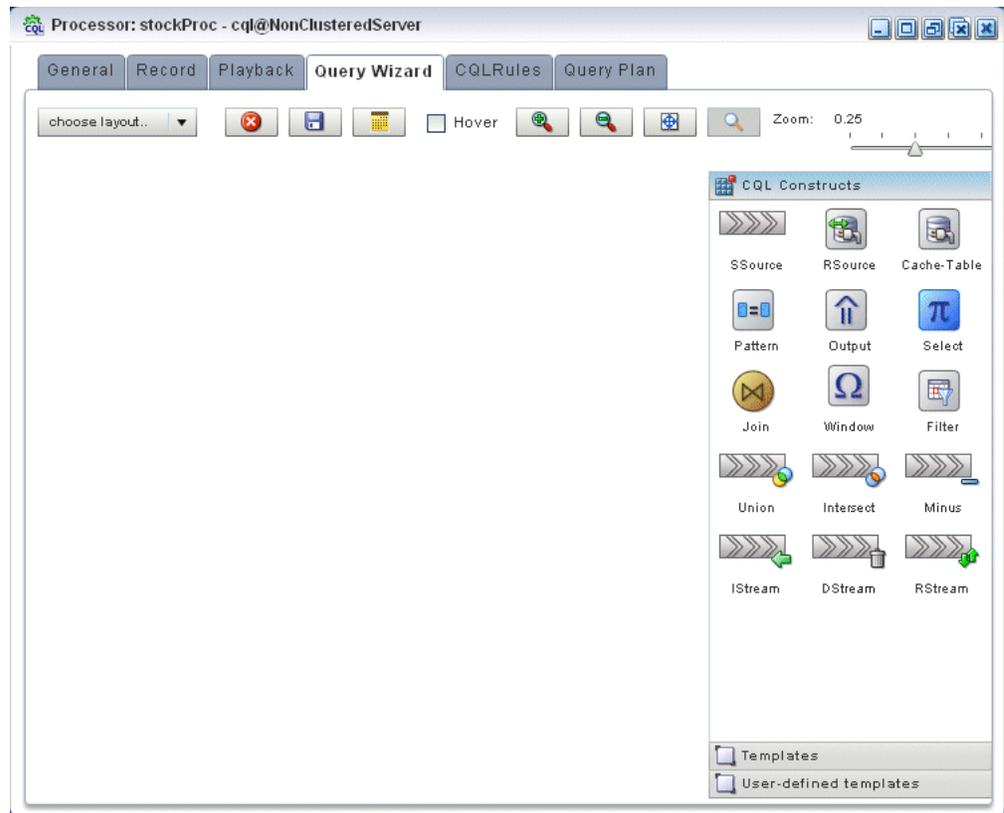
7. Double-click the **stockProcessor** Oracle CQL processor icon.
The Oracle CQL processor screen appears as [Figure 3–43](#) shows.

Figure 3–63 Oracle CQL Processor: General Tab



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

The Query Wizard screen appears as [Figure 3–44](#) shows.

Figure 3–64 Oracle CQL Processor: Query Wizard Tab

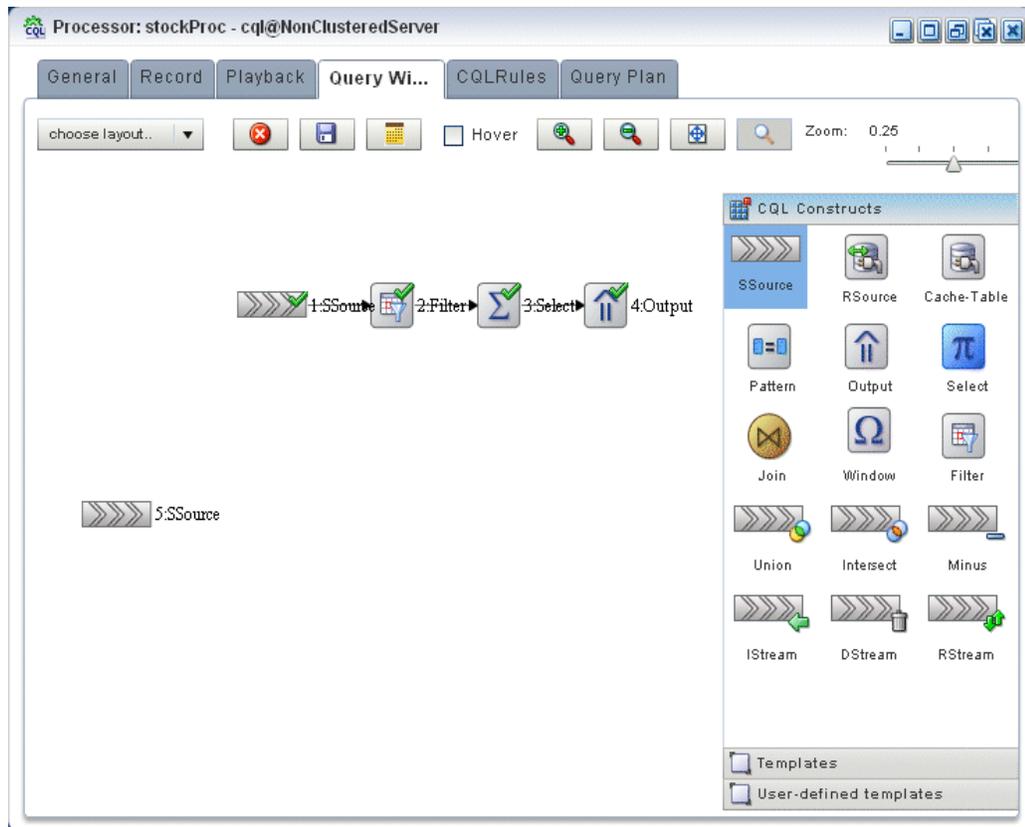
You can use the Oracle CQL Query Wizard to construct an Oracle CQL query from a template or from individual Oracle CQL constructs.

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

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

9. Click and drag an SSource icon (Stream Source) from the CQL Constructs palette and drop it anywhere in the Query Wizard canvas as [Figure 3–65](#) shows.

Figure 3–65 Query Wizard: SSource for Moving Average Query



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

The SSource configuration screen appears as [Figure 3–66](#) shows.

Figure 3–66 SSource Configuration Dialog: Moving Average Query

Stream [ID: 5]

Type Stream View

StockVolGt1000 AS

Properties (3)	
price	double
symbol	String
volume	long

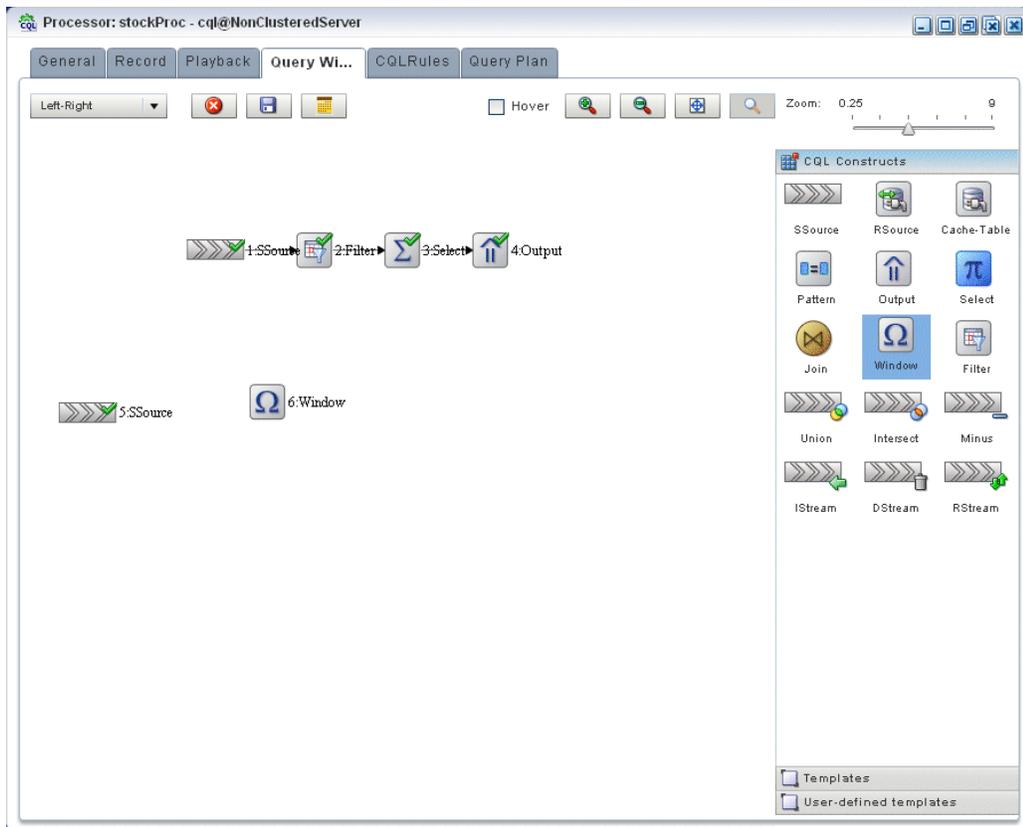
Generated CQL Statement

```
SELECT * FROM StockVolGt1000
```

Help Validate Save Cancel

11. Configure the SSource dialog as follows:
 - Select **View** as the **Type**.
 - Select the **StockVolGt1000** view from the **Select a source** pull-down menu.
12. Click **Save**.
13. Click **Save Query**.
14. Click and drag a Window icon from the CQL Constructs palette and drop it anywhere in the Query Wizard canvas as [Figure 3–67](#) shows.

Figure 3–67 Query Wizard: Window for Moving Average Query



15. Click on the **SSource** icon and drag to the **Window** icon to connect the Oracle CQL constructs.

16. Double-click the **Window** icon.

The SSource configuration screen appears as [Figure 3–66](#) shows.

Figure 3–68 Window Configuration Dialog: Moving Average Query

We want to create a sliding window over the last 2 events, partitioned by `symbol`.

17. Configure the Window dialog as follows:

- Select **symbol** in the **Source Property List** to add it to the **Partition List**.
- Select **Partition** as the **Type**.
- Select **Row Based** and enter 2 in the **Row Based** field.

18. Click Add Window.

The Query Wizard adds the sliding window to the Generated CQL Statement as [Figure 3–69](#) shows.

Figure 3–69 Window Configuration Dialog: After Adding Window

Window [ID : 6]

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

price
symbol

symbol

Type Now Time Row Partition Unbounded

Row Based 2

Time Based

ns micros ms sec min hour day

Slide Row Based

Time Based

ns micros ms sec min hour day

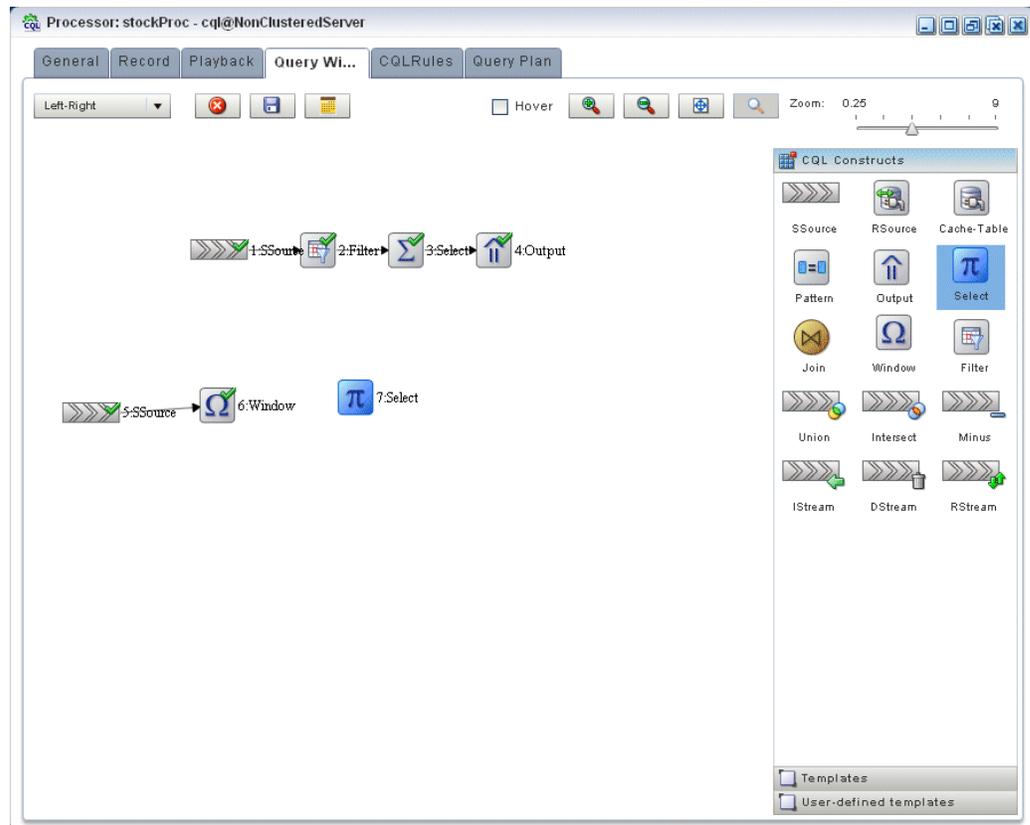
Generated CQL Statement

```
SELECT * FROM StockVolIGT1000 [PARTITION BY symbol ROWS 2]
```

Help Add Window Validate Save Cancel

19. Click **Save**.
20. Click **Save Query**.
21. Click and drag a Select icon from the CQL Constructs palette and drop it anywhere in the Query Wizard canvas as [Figure 3–67](#) shows.

Figure 3–70 Query Wizard: Select for Moving Average Query



22. Click on the **Window** icon and drag to the **Select** icon to connect the Oracle CQL constructs.

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

The Select configuration screen appears as [Figure 3–66](#) shows.

Figure 3–71 Select Configuration Dialog: Source Property symbol Selected

The screenshot shows a dialog window titled "Select [ID : 7]". It has four tabs: "Project", "Group", "Condition", and "Order". The "Project" tab is active, showing "Step 1- Project".

Options include:

- Distinct Results
- Target Event Type: **Select or Input Event Type** (dropdown)

Source Properties (select from here):

- StockVolGt1000 (dropdown)
- Properties (3):

price	double
symbol	String
volume	long

Source Properties (right panel):

- Select List (0)

Project Expression: AS (dropdown) (+)

Generated CQL Statement:

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

Buttons at the bottom: Help, Validate, Save, Cancel.

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

This is the source of moving average query: the view we created earlier (see "[To create a view source for the moving average query:](#)" on page 3-75).

25. Select **MovingAvgEvent** from the **Select or Input Event Type** pull-down menu.

This is the output event our moving average query will produced. We will map properties from the source events to this output event.

26. In the Source Properties list, select **symbol**.

The selected source property is added to the Project Expression as [Figure 3–72](#) shows.

Figure 3–72 Select Configuration Dialog: Source Property symbol Selected

The screenshot shows a dialog window titled "Select [ID : 7]" with a close button. It has four tabs: "Project", "Group", "Condition", and "Order". The "Project" tab is active, showing "Step 1- Project".

Options include:

- Distinct Results
- Target Event Type: **MovingAvgEvent** (dropdown)

Source Properties (select from here):

- StockVolGt1000 (dropdown)
- Properties (3):
 - price double
 - symbol String** (highlighted)
 - volume long

Source Properties (right panel):

- Select List (0)

Project Expression: StockVolGt1000.symbol

AS: **symbol** (dropdown)

Buttons: Select or Input Alias, +

Generated CQL Statement:

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

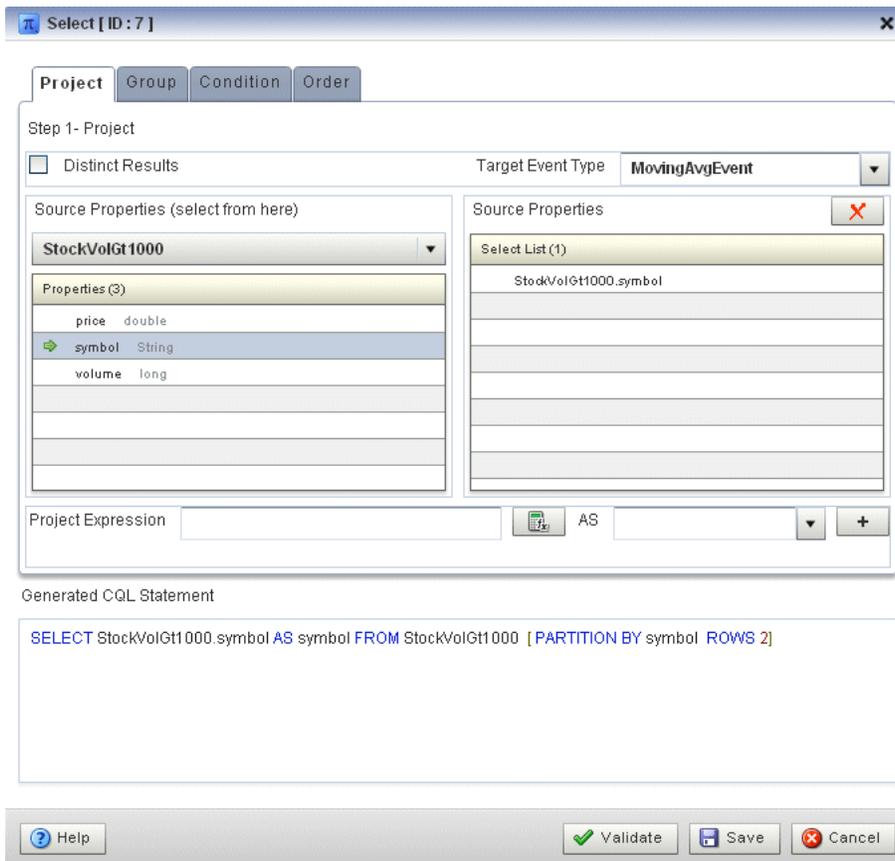
Footer buttons: Help, Validate, Save, Cancel.

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

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

The source property is added to the project expression of the Generated CQL Statement as [Figure 3–73](#) shows.

Figure 3–73 Select Configuration Dialog: Source Property symbol Mapped to Output Event Property



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

The selected source property is added to the Project Expression as [Figure 3–74](#) shows.

Figure 3–74 Select Configuration Dialog: Source Property price Selected

The screenshot shows a dialog window titled "Select [ID : 7]" with a close button (X). It has four tabs: "Project", "Group", "Condition", and "Order". The "Project" tab is active, showing "Step 1- Project".

Options include:

- Distinct Results
- Target Event Type: **MovingAvgEvent** (dropdown)

Source Properties (select from here):

- StockVolGt1000 (dropdown)
- Properties (3):
 - price double (selected with a green arrow)
 - symbol String
 - volume long

Source Properties (with a close button X):

- Select List (1):
 - StockVolGt1000.symbol

Project Expression: StockVolGt1000.price (with an Expression Builder icon and "AS" dropdown)

Generated CQL Statement:

```
SELECT StockVolGt1000.symbol AS symbol FROM StockVolGt1000 [PARTITION BY symbol ROWS 2]
```

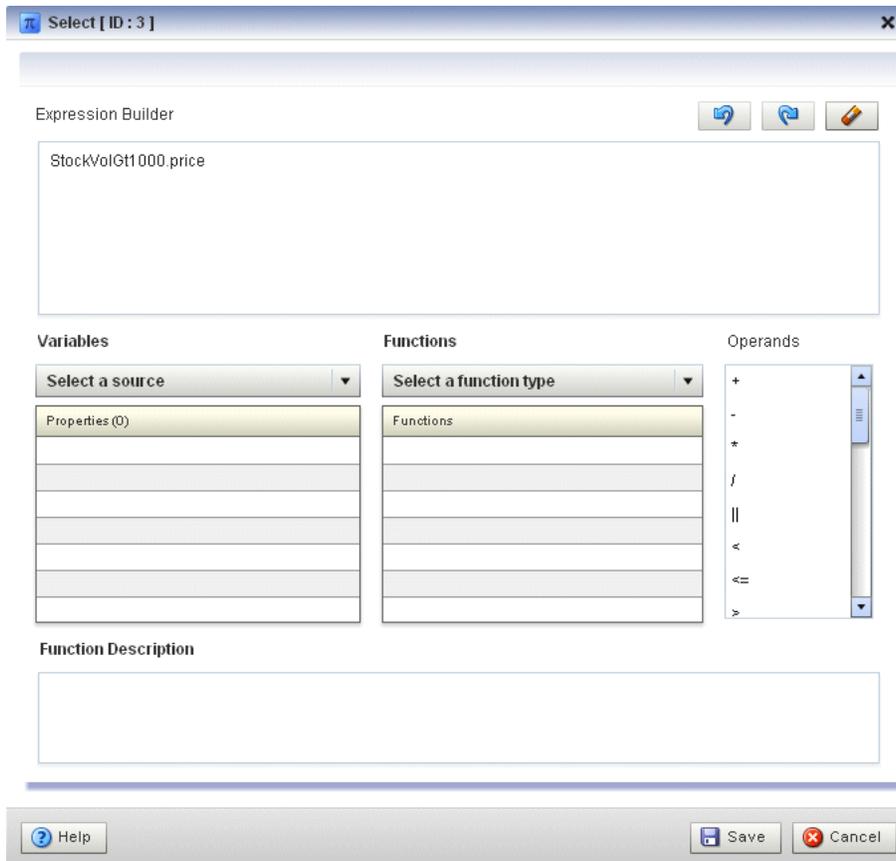
Buttons at the bottom: Help, Validate, Save, Cancel.

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

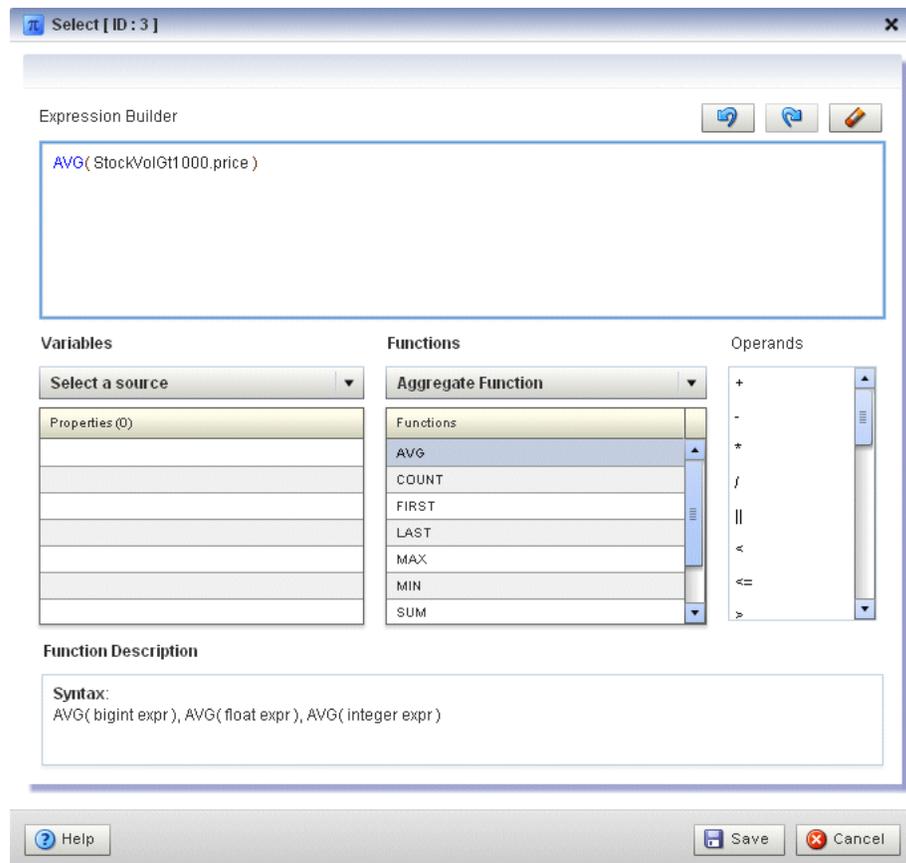
30. Click the Expression Builder button.

The Expression Builder dialog appears as [Figure 3–75](#) shows.

Figure 3–75 Expression Builder Dialog

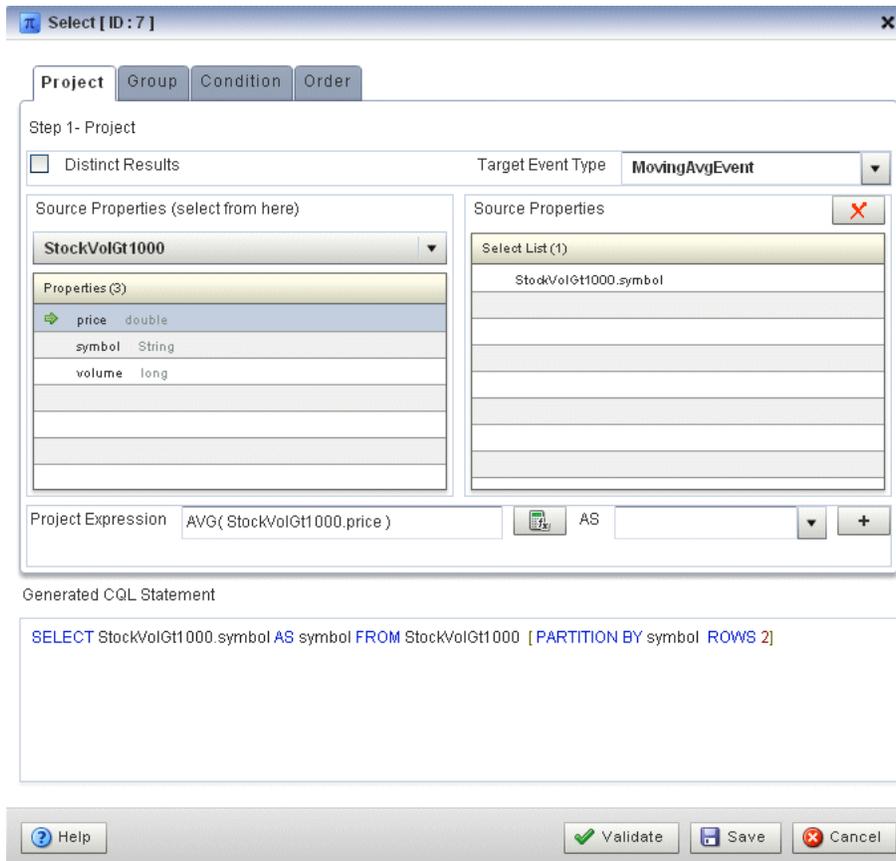


31. Select **Aggregate Functions** from the **Select a function type** pull-down menu.
 A list of the aggregate functions that Oracle CQL provides is displayed. We are going to use the **AVG** function.
32. Select the **StockVolGt1000.price** in the Expression Builder field.
33. Double-click the **AVG** function.
 The **AVG ()** function is wrapped around our selection in the Expression Builder field as [Figure 3–76](#) shows.

Figure 3–76 Expression Builder: Applying the AVG Function**34. Click Save.**

The expression is added to the Project Expression field as [Figure 3–77](#) shows.

Figure 3–77 Select Configuration Dialog: With Expression



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

36. Click the plus Sign button.

The source property is added to the project expression of the Generated CQL Statement as [Figure 3–78](#) shows.

Figure 3–78 Select Configuration Dialog: Source Property price Mapped to Output Event Property

37. Click Validate.

A validation error dialog is shown as [Figure 3–79](#) shows.

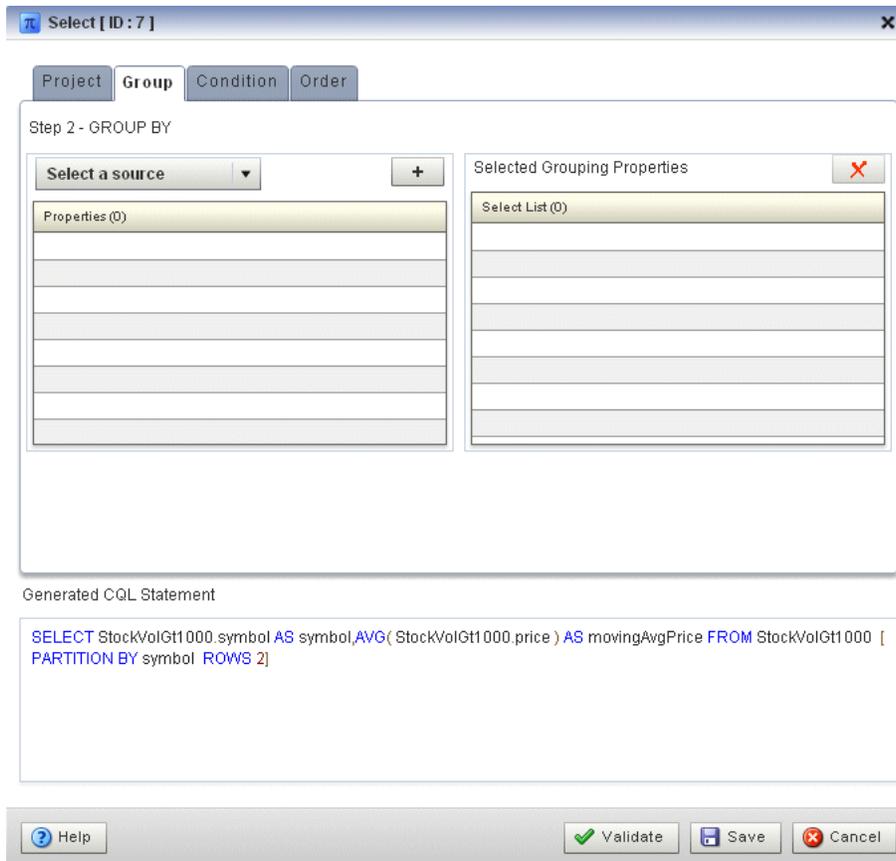
Figure 3–79 Validation Error: GROUP BY

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

38. Select the Group tab.

The Group tab appears as [Figure 3–80](#) shows.

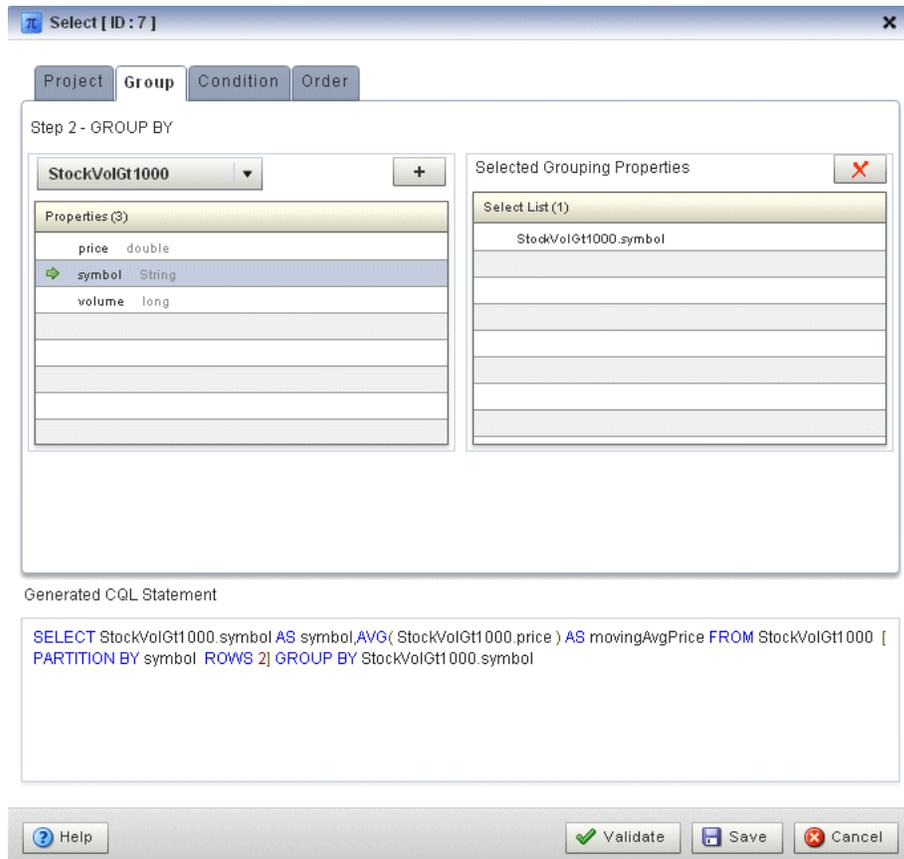
Figure 3–80 Group Tab



39. Configure the Group tab as follows:

- Select **StockVolGt1000** from the **Select a source** pull-down menu.
- Select **symbol** from the **Properties** list.
- Click the Plus Sign button.

The `symbol` property is added to `GROUP BY` clause as [Figure 3–81](#) shows.

Figure 3–81 Group Tab: With symbol Grouping Property

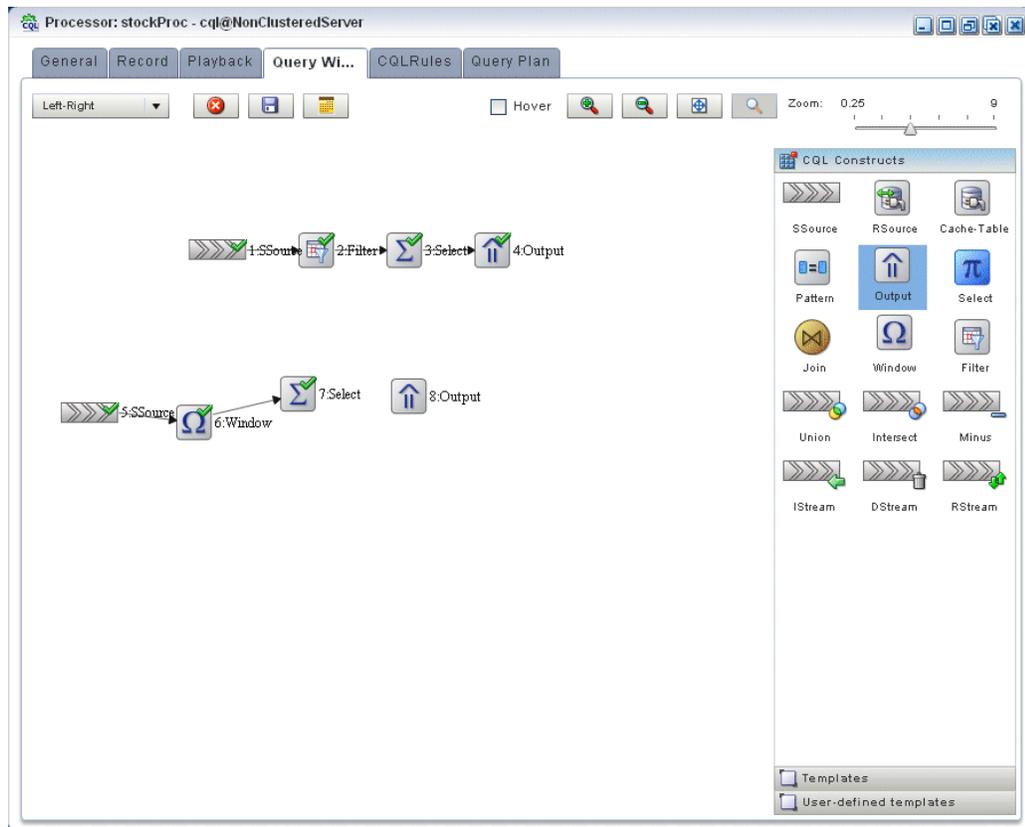
40. Click **Save**.

41. Click **Save Query**.

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

42. Click and drag an Output icon from the CQL Constructs palette and drop it anywhere in the Query Wizard canvas as [Figure 3–82](#) shows.

Figure 3–82 Query Wizard: Output



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

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

The Output configuration screen appears as [Figure 3–83](#) shows.

Figure 3–83 Output Configuration Dialog

The screenshot shows the 'Output [ID : 8]' dialog box. The 'Type' is set to 'Query'. The 'Query Name' is 'MovingAverage'. The 'Enable' checkbox is checked (true). The 'View' radio button is unselected. The 'View Name' and 'View Schema' fields are empty. The 'Project List' table contains two rows:

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

The 'Generated CQL Statement' is:

```
SELECT StockVolGt1000.symbol AS symbol,AVG( StockVolGt1000.price ) AS movingAvgPrice
FROM StockVolGt1000 [PARTITION BY symbol ROWS 2] GROUP BY StockVolGt1000.symbol
```

The dialog has buttons for Help, Inject Rule, Replace Rule, Validate, Save, and Cancel.

45. Configure the Output as follows:

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

46. Click **Inject Rule**.

The Inject Rule Confirmation dialog appears as [Figure 3–57](#) shows.

Figure 3–84 Inject Rule Confirmation Dialog

The screenshot shows a 'Confirmation' dialog box with the message 'Rule has been successfully added.' and an 'OK' button.

47. Click **OK**.

The Query Wizard adds the rule to the `cqlProc` processor.

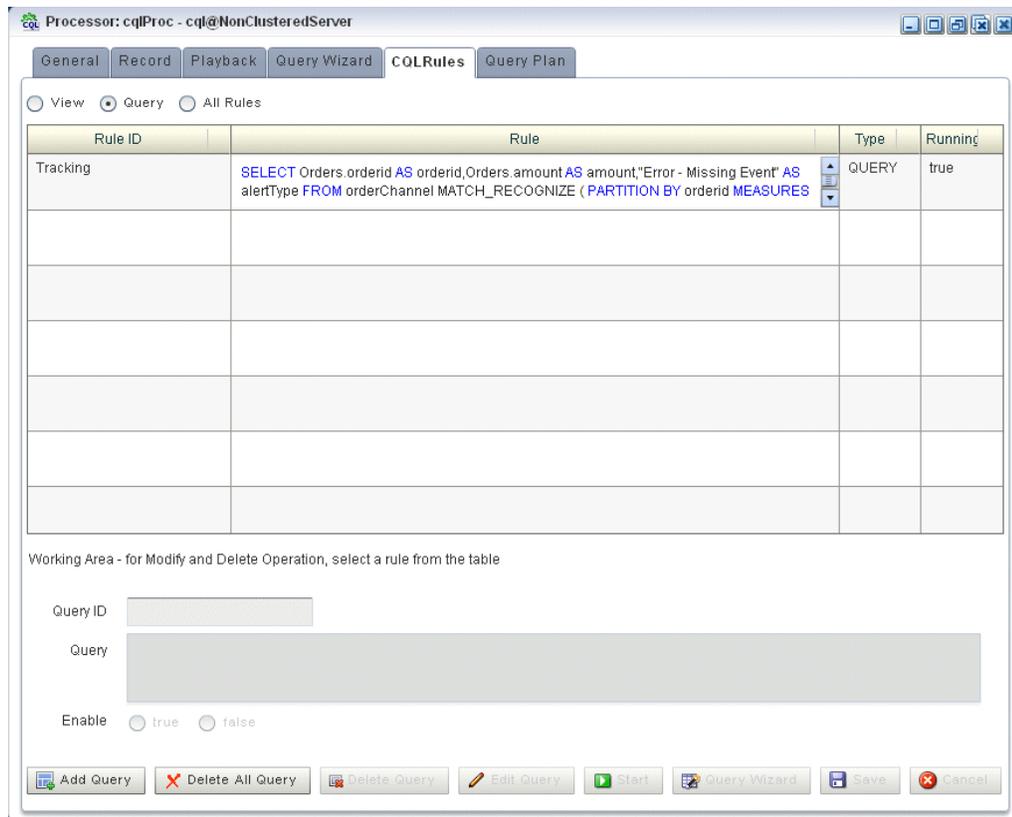
48. Click **Save**.

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

The CQL Rules tab appears as [Figure 3–58](#) shows.

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

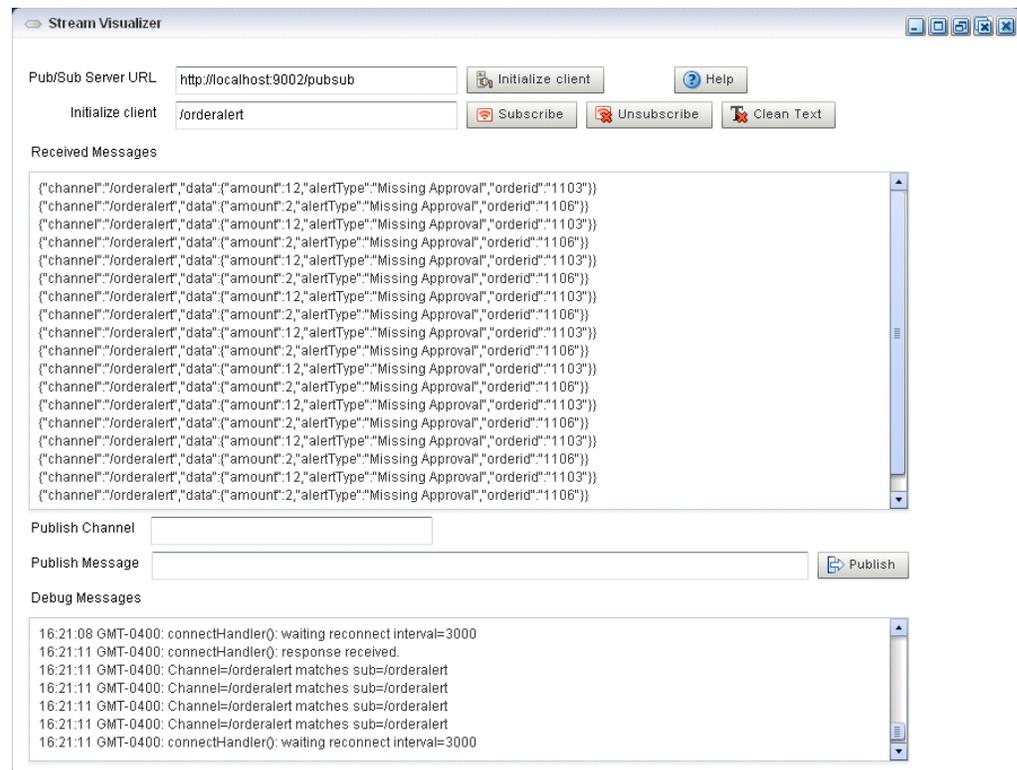
Confirm that your `MovingAverage` query is present.

Figure 3–85 CQL Rules Tab With View MovingAverage**To test the moving average query:**

1. To simulate the data feed for the moving average query, open a new command window and set your environment as described in [Section 3.5, "Setting Your Development Environment."](#)
2. 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`.
3. Run the load generator using the `stockData.prop` properties file:
 - a. On Windows:


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


```
prompt> runloadgen.sh stockData.prop
```
4. In the Oracle CEP Visualizer, click the **ViewStream** button in the top pane. The Stream Visualizer screen appears as [Figure 3–38](#) shows.

Figure 3–86 Stream Visualizer: Showing Moving Average Query Output

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

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

Glossary

Adapter

An element of the [EPN](#) that interfaces directly to an inbound event source. Adapters understand the inbound protocol, and are responsible for converting the event data into a normalized form that can be queried by a [POJO](#). Adapters forward the normalized event data into a [Stream](#).

Aggregate Function

Aggregate functions return a single aggregate result based on group of tuples, rather than on a single tuple.

See also [Function](#) and [Single-Row Function](#).

CEP

Complex Event Processing.

Channel

A channel represents the physical conduit through which events flow between other types of components, such as between an [Adapter](#) and a [Processor](#), and between a [Processor](#) and an [Event Bean](#). A channel can model a [Stream](#) or [Relation](#).

Condition

An Oracle CQL condition specifies a combination of one or more expressions and logical (Boolean) operators and returns a value of TRUE, FALSE, or UNKNOWN.

Constant value

A fixed data value. Synonymous with [Literal](#).

CQL

Oracle Continuous Query Language. Supersedes [EDA](#).

Data Feed

A synonym for [Event Source](#).

Destination

An Oracle CQL destination identifies a consumer of query results such as the Enterprise Link BAM Adapter, JMS queue or topic, or file. You can alter a query to add a destination.

Deterministic Garbage Collection

Short, predictable pause times for memory heap garbage collection, which is the process of clearing dead objects from the heap, thus releasing that space for new objects.

DStream

A relation-to-stream operator that represents deleted tuples.

EDA

Event-Driven Architecture.

EPL

Oracle Event Processing Language. Superseded by [CQL](#).

EPN

Oracle Event Processing Network. An EPN is the arbitrary interconnection of [Adapter](#), [Stream](#), [POJO](#), and business logic POJOs used by Oracle CEP to process events.

Event Bean

A [POJO](#) to that contains the business logic executed when a notable event is detected. An event bean is an [Event Sink](#).

Event Rule

A query, expressed in [CQL](#) or [EDA](#), executed by a [POJO](#) to filter and aggregate events.

Event Sink

A component that consumes events, such as a [Processor](#).

See also [Event Source](#).

Event Source

A component that provides events, such as a sensor, wire service, or stock ticker.

See also [Data Feed](#) and [Event Sink](#).

Expressions

An Oracle CQL expression is a combination of one or more values, operators, and Oracle CQL functions that evaluates to a value. An expression generally assumes the datatype of its components.

See also [Condition](#) and [Function](#).

Format model

A character **literal** that describes the format of datetime or numeric data stored in a character string.

Function

Oracle CQL functions are similar to operators in that they manipulate data items and return a result. Functions differ from operators in the format of their arguments. This format enables them to operate on zero, one, two, or more arguments.

See also [Condition](#), [Aggregate Function](#), and [Single-Row Function](#).

Incremental Processing

A user-defined aggregate function design pattern that improves scalability and performance by ensuring that the cost of (re)computation on arrival of new events will be proportional to the number of new events as opposed to the total number of events seen thus far.

If your user-defined aggregate function supports incremental processing, you specify the `supports incremental processing` clause in the `register function` statement to instruct the Oracle CEP Service Engine to supply only the new event data as opposed to performing a rescan over already processed event data.

IStream

A relation-to-stream operator that represents inserted tuples.

Join

A query that combines rows from two or more streams, views, or relations.

Latency

An expression of how much time it takes for data to get from one designated point to another.

Literal

A fixed data value. Synonymous with [Constant value](#).

Monotonic

A relation R is monotonic if and only if $R(t_1)$ is contained in and equal to $R(t_2)$ whenever $t_1 \leq t_2$. A sequence of values increases monotonically if and only if they never decrease. Conversely, a sequence decreases monotonically if and only if they never increase.

Now window

A special case of the time-based sliding window on a stream S that takes a time-interval T as a parameter and is specified by: S [Range T]. A Now window is defined as: S [Now] (short for S [Range 0]). When $T = 0$, the relation at time t consists of tuples obtained from elements of S with timestamp t .

See also [Sliding window](#).

Operators

Oracle CQL operators manipulate data items and return a result. Syntactically, an operator appears before or after an operand or between two operands.

OSGi

A dynamic module system for Java that provides a service-oriented, component-based environment and standardized software lifecycle management. Oracle CEP applications are packaged and deployed as OSGi bundles. For more information, see <http://www.osgi.org/>.

Partitioned window

A partitioned sliding window on a stream S takes a positive integer number of tuples N and a subset $\{A_1, \dots, A_k\}$ of the stream's attributes as parameters and is specified by: S [Partition By $A_1 \dots A_k$ Rows N] or, optionally, S [Partition By $A_1 \dots A_k$ Rows N Range T].

See also [Sliding window](#).

POJO

A Plain Old Java Object. A Java class that is not required to implement a third-party interface or extend a third-party class. In Oracle CEP, you can express your business logic using POJOs.

Processor

An element of the [EPN](#) that consumes normalized event data from a stream, processes it using queries (expressed in [CQL](#) or [EDA](#)), and may generate new events to an output stream.

Query

A query is an operation that retrieves data from one or more streams or views. In this reference, a top-level `SELECT` statement is called a **query**.

Real-time

A level of computer responsiveness that a user senses as sufficiently immediate or that enables the computer to keep up with some external process (for example, to present visualizations of the weather as it constantly changes).

Relation

A relation is time-varying bag of tuples. Here "time" refers to an instant in the time domain. 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. The updates are required to arrive at the system in the order of increasing timestamps. Like streams, relations have a fixed schema to which all tuples conform.

RStream

A relation-to-stream operator that maintains the entire current state of its input relation and outputs all of the tuples as insertions at each time step.

Single-Row Function

Single-row functions return a single result row for every row of a queried stream or view.

See also [Function](#) and [Aggregate Function](#).

Sliding window

A stream-to-relation operator based on the window specification derived from SQL99.

See also: [Now window](#), [Partitioned window](#), [Unbounded window](#), [tuple-based](#), and [Unbounded window](#), [time-based](#).

Source

An Oracle CQL source identifies a producer of data that a Oracle CQL query operates on such as the Enterprise Link BAM Adapter, JMS queue or topic, or file. You can alter a stream or relation to add a data source.

Spring Framework

A light-weight, open source application framework for Java. Oracle CEP server uses the Spring Framework to host Oracle CEP applications. For more information, see <http://www.springframework.org/>.

Stream

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. A stream has an associated schema consisting of a set of named attributes, and all tuples of the stream conform to the schema.

A stream is a bag of tuple-timestamp pairs, which can be represented as a sequence of timestamped tuple "insertions".

In Oracle CEP, a stream is modeled as a channel component.

See also [Tuple](#) and [Channel](#).

Throughput

An Oracle CQL source identifies a producer of data that a Oracle CQL query operates on such as the Enterprise Link BAM Adapter, JMS queue or topic, or file. You can alter a stream or relation to add a data source.

Tuple

The term "tuple of a stream" denotes the ordered list of data (excluding timestamp data) portion of a stream element (the s of $\langle s, t \rangle$). For example, a stock ticker data stream might appear like this where each stream element is made up of $\langle \text{timestamp value}, \text{stock symbol}, \text{stock price} \rangle$:

```
...
<timestampN>    NVDA, 4
<timestampN+1>  ORCL, 62
<timestampN+2>  PCAR, 38
<timestampN+3>  SPOT, 53
<timestampN+4>  PDCO, 44
<timestampN+5>  PTEN, 50
...
```

In the stream element $\langle \text{timestampN+1} \rangle$ ORCL, 62, the tuple is ORCL, 62.

See also [Stream](#).

Unbounded window, time-based

A special case of the time-based sliding window on a stream S that takes a time-interval T as a parameter and is specified by: S [Range T]. An Unbounded window is defined as: S [Range Unbounded] (short for S [Range infinity]). When $T = \text{infinity}$, the relation at time t consists of tuples obtained from all elements of S up to t .

See also [Sliding window](#).

Unbounded window, tuple-based

A special case of the tuple-based sliding window on a stream S that takes a number of tuples N as a parameter and is specified by: S [Rows N]. An Unbounded window is defined as: S [Rows Unbounded] (short for S [Rows infinity] and equivalent to S [Range Unbounded]). When $T = \text{infinity}$, the relation at time t consists of tuples obtained from all elements of S up to t .

See also [Sliding window](#).

View

An Oracle CQL view represents an alternative selection on a stream or relation. In Oracle CQL, you use a view instead of a subquery. A top-level VIEW statement that you create using the [REGISTER | CREATE] VIEW statement is called a **view**.

Symbols

- *
 - maximal pattern quantifier (0 or more times), 3-60
- *?
 - minimal pattern quantifier (0 or more times), 3-60
- +
 - maximal pattern quantifier (1 or more times), 3-60
- +?
 - minimal pattern quantifier (1 or more times), 3-60
- ?
 - maximal pattern quantifier (0 or 1 time), 3-60
- ??
 - minimal pattern quantifier (0 or 1 time), 3-60

A

- adapters, 1-3
- Ant
 - CQL sample targets, 3-50
 - Event Record and Playback sample targets, 3-45
 - FX sample targets, 3-17
 - HelloWorld sample targets, 3-9
 - installing, 2-1
 - Signal Generation sample targets, 3-30

B

- beans, 1-3

C

- channels, 1-3
- CQL
 - sample
 - about, 3-47
- CQL sample
 - Ant targets, 3-50
 - building, 3-49
 - deploying, 3-49
 - running, 3-48

D

- development environment, 3-4

E

- EDA
 - decoupled components, 1-2
 - event sinks, 1-2
 - event sources, 1-2
 - event streams, 1-2
 - example, 1-1
 - rule driven, 1-2
- environment, 3-4
- EPN
 - about, 1-3
 - FX sample assembly file, 3-19
 - HelloWorld sample assembly file, 3-11
 - Signal Generation sample assembly file, 3-31
 - topologies, 1-3
- event beans, 1-3
- Event Processing Network. *See* EPN
- Event Record and Playback sample
 - about, 3-36
 - Ant targets, 3-45
 - building, 3-44
 - deploying, 3-44
 - implementation, 3-46
 - running, 3-37
- event sinks, 1-2
- event sources, 1-2
- event streams, 1-2
- Event-Driven Architecture. *See* EDA

F

- FX sample
 - about, 3-14
 - Ant targets, 3-17
 - building, 3-16
 - component configuration file, 3-22
 - deploying, 3-16
 - EPN assembly file, 3-19
 - implementation, 3-17
 - running, 3-15

H

- HelloWorld sample
 - about, 3-6

- Ant targets, 3-9
- building, 3-8
- component configuration file, 3-13
- deploying, 3-8
- EPN assembly file, 3-11
- implementation, 3-9
- running, 3-7

I

- installation
 - Ant, 2-1
 - file name, 1-6, 2-2
 - installation programs, 1-6, 2-2
 - installers, 1-6, 2-2
 - JRockit, 2-1
 - mode
 - console, 2-2
 - graphical, 2-2
 - silent, 2-2
 - patches, 2-10
 - post-installation tasks, 2-11
 - pre-installation tasks, 2-1
 - samples, 3-3

J

- JRockit
 - installing, 2-1

M

- MATCH_RECOGNIZE
 - MEASURES clause, 3-67
- MEASURES clause, 3-67

O

- Oracle CEP Visualizer
 - samples, 3-3

P

- patches, 2-10
- platforms, 1-6
- processors, 1-3

S

- samples
 - about, 3-1
 - Ant targets
 - CQL, 3-50
 - Event Record and Playback, 3-45
 - FX, 3-17
 - HelloWorld, 3-9
 - Signal Generation, 3-30
 - building
 - CQL, 3-49
 - Event Record and Playback, 3-44
 - FX, 3-16

- HelloWorld, 3-8
- Signal Generation, 3-29
- component configuration file
 - FX, 3-22
 - HelloWorld, 3-13
 - Signal Generation, 3-34
- deploying
 - CQL, 3-49
 - Event Record and Playback, 3-44
 - FX, 3-16
 - HelloWorld, 3-8
 - Signal Generation, 3-29
- development environment, 3-4
- EPN assembly file
 - FX, 3-19
 - HelloWorld, 3-11
 - Signal Generation, 3-31
- implementation
 - Event Record and Playback, 3-46
 - FX, 3-17
 - HelloWorld, 3-9
 - Signal Generation, 3-30
- installing, 3-3
- Oracle CEP Visualizer, 3-3
- performance, 3-3
- ready-to-run, 3-2
- running
 - CQL, 3-48
 - Event Record and Playback, 3-37
 - FX, 3-15
 - HelloWorld, 3-7
 - Signal Generation, 3-27
- source, 3-2
- Signal Generation sample
 - about, 3-26
 - Ant targets, 3-30
 - building, 3-29
 - component configuration file, 3-34
 - deploying, 3-29
 - EPN assembly file, 3-31
 - implementation, 3-30
 - running, 3-27
- supported platforms, 1-6

T

- topologies, 1-3

U

- user code, 1-3