Oracle® Fusion Middleware

Administrator's Guide for Oracle Event Processing

11*g* Release 1 (11.1.1.9)

E14300-11

February 2015

Documentation for administrators that describes how to manage Oracle Event Processing applications and clusters, including deploying and administering, as well as configuring use of Oracle Coherence, Jetty, JMX, JDBC, and security tools.

ORACLE

Oracle Fusion Middleware Administrator's Guide for Oracle Event Processing 11*g* Release 1 (11.1.1.9)

E14300-11

Primary Author: Oracle Corporation

Contents

10 Configuring Security for Oracle Event Processing

11 Configuring Jetty for Oracle Event Processing

12 Configuring JMX for Oracle Event Processing

13 Configuring JDBC for Oracle Event Processing

14 Configuring HTTP Publish-Subscribe for Oracle Event Processing

15 Configuring Logging and Debugging for Oracle Event Processing

B Deployer Command-Line Reference

C Security Utilities Command-Line Reference

List of Tables

List of Examples

List of Figures

Preface

This document describes how to configure and manage the Oracle Event Processing server.

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

Audience

This document is intended for all users of Oracle Event Processing.

Documentation Accessibility

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

Access to Oracle Support

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

Related Documents

For more information, see the following:

- *Oracle Fusion Middleware Getting Started Guide for Oracle Event Processing*
- *Oracle Fusion Middleware Developer's Guide for Oracle Event Processing for Eclipse*
- *Oracle Fusion Middleware Visualizer User's Guide for Oracle Event Processing*
- *Oracle Fusion Middleware Java API Reference for Oracle Event Processing*

- *Oracle Fusion Middleware CQL Language Reference for Oracle Event Processing*
- *Oracle Fusion Middleware EPL Language Reference for Oracle Event Processing*
- *Oracle Database SQL Language Reference* at http://download.oracle.com/docs/cd/B28359_01/server.111/b28286/toc.htm
- SQL99 Specifications (ISO/IEC 9075-1:1999, ISO/IEC 9075-2:1999, ISO/IEC 9075-3:1999, and ISO/IEC 9075-4:1999)
- Oracle Event Processing Forum: http://forums.oracle.com/forums/forum.jspa?forumID=820

Conventions

The following text conventions are used in this document:

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

What's New in This Guide

This guide has been updated in several ways. The following table lists the sections that have been added or changed.

For a list of known issues (release notes), see the "Known Issues for Oracle SOA Products and Oracle AIA Foundation Pack" at http://www.oracle.com/technetwork/middleware/docs/soa-aiafp-knownissuesindex-364630.html.

Sections	Changes Made	February 2013
Chapter 9: Configuring Network I/O for Oracle Event Processing		
Section , "Network I/O Providers"	Removed reference to several classes that are for internal use only.	X
Appendix B: Deployer Command Line Reference		
Section , "Deployment Commands"	Removed startLevel command, which is not supported.	X

Part I

Overview

Part I contains the following chapter:

- Chapter 1, "Overview of Oracle Event Processing Server Administration"

1

Overview of Oracle Event Processing Server Administration

This chapter provides an overview of administering Oracle Event Processing, introducing Oracle Event Processing servers and domains and server lifecycle, as well as server configuration, administration tools and tasks.

This chapter includes the following sections:

Understanding Oracle Event Processing Servers and Domains

An Oracle Event Processing *server* consists of logically related resources and services to which you deploy Oracle Event Processing applications. Services include:

- Network Input/Output (I/O)—server and client Internet Protocol (IP) port access, IPv4 and IPv6 support, and a variety of blocking and non-blocking network I/O providers.
- Security—security services such as SSL, password stores, and authentication and authorization providers.
- Jetty—HTTP server.
- Java Management Extensions (JMX)—to provide programmatic access to Oracle Event Processing server and application behavior.
- JDBC datasources—to access relational databases to store events for event record and playback, to access a table as an event source for Oracle CQL queries, and to join a table in EPL queries.
- HTTP publish-subscribe server—to push event messages to subscribed clients such as the Oracle Event Processing Visualizer and your own Web 2.0 applications.
- logging—to support monitoring and troubleshooting server and application operation.

All the files that apply to a server are contained in a single server directory. The main configuration file for the server is called `config.xml`—this is where you configure the server's services and specify to which domain the server belongs. For more

information, see Section , "Oracle Event Processing Server Configuration Files".

An Oracle Event Processing *domain* is the management unit of a set of one or more servers. There are two types of domain:

- Standalone-server domain—A domain that contains a single server. This is the type of domain created by default by the Configuration Wizard and is the starting point for a multi-server domain.

  For more information, see Chapter 3, "Administering Oracle Event Processing Standalone-Server Domains".

- Multi-server domain—A domain that contains two or more servers that share the same multicast address and port and share a security provider. Multi-server domains enable high availability for Oracle Event Processing applications. When you deploy an application to a multi-server domain, the application is replicated to each server in the domain.

  The servers in a multi-server domain can be located on the same computer or on separate computers; what ties the servers together in a multi-server domain is that they have the same multicast address and port and belong to the same domain, all of which are configured in the server's `config.xml` file.

  For more information, see:

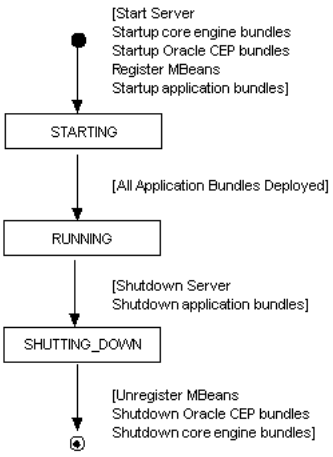
  – Chapter 5, "Administering Multi-Server Domains With Oracle Coherence"

  – Chapter 7, "Administering Multi-Server Domains With Oracle Event Processing Native Clustering"

Understanding Oracle Event Processing Server Lifecycle

Figure 1–1 shows a state diagram for the Oracle Event Processing server lifecycle. In this diagram, the state names (`STARTING`, `RUNNING`, and `SHUTTING_DOWN`) correspond to the `ServerRuntimeMBean` method `getState()` return values. These states are specific to Oracle Event Processing; they are not OSGi bundle states.

Figure 1–1 Oracle Event Processing Server Lifecycle State Diagram



Note: For information on Oracle Event Processing application lifecycle, see "Oracle Event Processing Application Lifecycle" in the *Oracle Fusion Middleware Developer's Guide for Oracle Event Processing for Eclipse*.

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

- Section , "User Action: Start Oracle Event Processing Server"
- Section , "User Action: Stop Oracle Event Processing Server"

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

- "Oracle Event Processing APIs" in the *Oracle Fusion Middleware Developer's Guide for Oracle Event Processing for Eclipse*
- *Oracle Fusion Middleware Java API Reference for Oracle Event Processing*

User Action: Start Oracle Event Processing Server

After you start the Oracle Event Processing server, it performs the following actions:

1. Starts core engine bundles.
2. Starts Oracle Event Processing bundles.
3. Registers MBeans.
4. Oracle Event Processing server state is now `STARTING`.
5. Starts application bundles.
6. Oracle Event Processing server state is now `RUNNING`.

User Action: Stop Oracle Event Processing Server

After you shutdown the Oracle Event Processing server, it performs the following actions:

1. Oracle Event Processing server state is `SHUTTING_DOWN`.
2. Unregister `ServerRuntimeMBean`.

   Oracle Event Processing server ceases to have a state.
3. Shuts down Oracle Event Processing bundles.
4. Shuts down application bundles.
5. Shuts down core engine bundles.

Understanding Oracle Event Processing Server Configuration

Oracle Event Processing server configuration falls into two categories:

- configuring a server
- configuring the applications you deploy to a server

For each configuration category, you can perform the configuration task:

- Statically: by editing an XML file manually.

  Using this approach, you must restart the server or redeploy the application after making a change.

- Dynamically: by manipulating management beans (MBeans) using Oracle Event Processing Visualizer, `wlevs.Admin` command-line utility, or programmatically using JMX,

  Using this approach, you do not have to restart the server or redeploy the application after making a change

You configure the server statically by:

1. Stopping the Oracle Event Processing server.
2. Editing the Oracle Event Processing server `config.xml` file located in the server's domain directory
3. Starting the Oracle Event Processing server.

There are some server configuration tasks that you can only perform statically, such as configuring Jetty.

There are some server configuration tasks that you can perform dynamically using management beans (MBeans). In this case you do not have to manually stop and start the server for the changes to take effect.

You can dynamically configure Oracle Event Processing servers and applications using Oracle Event Processing Visualizer, `wlevs.Admin` command line utility, or your own Java code using Oracle Event Processing standards-based interfaces that are fully compliant with JMX APIs (`http://java.sun.com/javase/technologies/core/mntr-mgmt/javamanagement/`).

Typically, you statically configure an application when you initially create it, as described in *Oracle Fusion Middleware Developer's Guide for Oracle Event Processing for Eclipse*. In particular, you configure the event processing network (EPN) of the application by creating the EPN assembly file, and configure individual components of the application (adapters, channels, and processors) by creating their configuration files.

After you deploy an application, you can dynamically change its configuration, as well as the configuration of its individual components, by manipulating the MBeans that the Oracle Event Processing server automatically creates for the application and its components. A typical task is to dynamically configure the Oracle CQL rules for the processors of a deployed application. You do this using Oracle Event Processing Visualizer, wlevs.Admin command-line utility, or JMX.

This section describes:

- Section , "Oracle Event Processing Server Configuration Files"
- Section , "Configuring an Oracle Event Processing Server by Manually Editing the config.xml File"
- Section , "Configuration History Management"
- Section , "Configuring the Oracle Event Processing Server bootclasspath"

For more information, see:

- Section , "Configuring Oracle Event Processing Servers"
- *Oracle Fusion Middleware Visualizer User's Guide for Oracle Event Processing*

- Appendix A, "wlevs.Admin Command-Line Reference"
- Chapter 12, "Configuring JMX for Oracle Event Processing"

Oracle Event Processing Server Configuration Files

By default, the Configuration Wizard creates domains in the *ORACLE_CEP_HOME*/user_projects/domains directory, where *ORACLE_CEP_HOME* refers to the Oracle Event Processing installation directory such as d:\oracle_cep.

For example, d:\oracle_cep\user_projects\domains\my_domain, where my_domain is the a domain directory. Each Oracle Event Processing server you create in this domain will have a subdirectory in my_domain such as c:\oracle_cep\user_projects\domains\my_domain\server1.

The following list describes the important files and directories of a server in a domain, relative to the server directory (such as c:\oracle_cep\user_projects\domains\my_domain\server1):

- deployments.xml—XML file that contains the list of applications, packaged as OSGi bundles, that are currently deployed to the Oracle Event Processing instance of this domain. You never update this file manually to deploy applications, but rather, use the deployer tool.
- startwlevs.cmd—Command file used to start an instance of Oracle Event Processing. The UNIX equivalent is called startwlevs.sh.
- stopwlevs.cmd—Command file used to stop an instance of Oracle Event Processing. The UNIX equivalent is called stopwlevs.sh.
- config/config.xml—XML file that describes the services that have been configured for the Oracle Event Processing instance. Services include logging, debugging, Jetty Web Service, and JDBC data sources.

  For more information, see Section , "Configuring an Oracle Event Processing Server by Manually Editing the config.xml File".
- config/security*—Files that configure security for the domain.
- config/atnstore.txt—File that lists the configured users and groups for this domain.

Configuring an Oracle Event Processing Server by Manually Editing the config.xml File

The most efficient, least error-prone way to configure an Oracle Event Processing server is to use one or more of the Oracle Event Processing administration tools as Section , "Understanding Oracle Event Processing Server Administration Tools" describes.

Optionally, you can perform Oracle Event Processing server configuration by manually editing the Oracle Event Processing server config.xml file. For more information on the location of the Oracle Event Processing server config.xml file, see Section , "Oracle Event Processing Server Configuration Files".

Caution: If you update the config.xml file manually to change the configuration of an Oracle Event Processing server, you must restart the server for the change to take effect.

You can configure the following server objects and features using the config.xml file (the referenced sections describe the exact elements you must add or update):

- How the servers in a multi-server domain are configured together. This includes the multicast address and port, the groups, and so on.

  See:

  - Chapter 3, "Administering Oracle Event Processing Standalone-Server Domains"
  - Chapter 5, "Administering Multi-Server Domains With Oracle Coherence"
  - Chapter 7, "Administering Multi-Server Domains With Oracle Event Processing Native Clustering"
- Network I/O.

  See Chapter 9, "Configuring Network I/O for Oracle Event Processing."
- Security.

  See Chapter 10, "Configuring Security for Oracle Event Processing."
- Jetty, an open-source, standards-based, full-featured Java Web Server.

  See Chapter 11, "Configuring Jetty for Oracle Event Processing."
- JMX, required to use the Oracle Event Processing Visualizer, `wlevs.Admin` utility, and Deployer utility

  See Chapter 12, "Configuring JMX for Oracle Event Processing."
- JDBC data source, used to connect to a relational database.

  See Chapter 13, "Configuring JDBC for Oracle Event Processing."
- HTTP publish-subscribe server.

  See Chapter 14, "Configuring HTTP Publish-Subscribe for Oracle Event Processing."
- Logging and debugging properties of the server. By default, the log level is set to `NOTICE`.

  See Chapter 15, "Configuring Logging and Debugging for Oracle Event Processing."

Example 1–1 shows a sample `config.xml`, from the *`ORACLE_CEP_HOME`*`/user_projects/domains/ocep_domain/defaultserver` template domain, shows how to configure some of these services.

Example 1–1 Sample Oracle Event Processing Server config.xml File

```
<?xml version="1.0" encoding="UTF-8"?>
<!--Sample XML file generated by XMLSpy v2007 sp2 (http://www.altova.com)-->
<n1:config
  xsi:schemaLocation="http://www.bea.com/ns/wlevs/config/server wlevs_server_config.xsd"
  xmlns:n1="http://www.bea.com/ns/wlevs/config/server"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
   <netio>
      <name>NetIO</name>
      <port>9002</port>
   </netio>
   <netio>
      <name>sslNetIo</name>
      <ssl-config-bean-name>sslConfig</ssl-config-bean-name>
      <port>9003</port>
   </netio>
   <work-manager>
      <name>JettyWorkManager</name>
```

```
   <min-threads-constraint>5</min-threads-constraint>
   <max-threads-constraint>10</max-threads-constraint>
</work-manager>
<jetty>
   <name>JettyServer</name>
   <network-io-name>NetIO</network-io-name>
   <work-manager-name>JettyWorkManager</work-manager-name>
   <secure-network-io-name>sslNetIo</secure-network-io-name>
</jetty>
<rmi>
   <name>RMI</name>
   <http-service-name>JettyServer</http-service-name>
</rmi>
<jndi-context>
   <name>JNDI</name>
</jndi-context>
<exported-jndi-context>
   <name>exportedJndi</name>
   <rmi-service-name>RMI</rmi-service-name>
</exported-jndi-context>
<jmx>
   <rmi-service-name>RMI</rmi-service-name>
   <jndi-service-name>JNDI</jndi-service-name>
</jmx>
<ssl>
   <name>sslConfig</name>
   <key-store>./ssl/evsidentity.jks</key-store>
   <key-store-pass>
      <password>{Salted-3DES}j4XEtuXmmvEl4M/NInwq0A==</password>
   </key-store-pass>
   <key-store-alias>evsidentity</key-store-alias>
   <key-manager-algorithm>SunX509</key-manager-algorithm>
   <ssl-protocol>TLS</ssl-protocol>
   <enforce-fips>false</enforce-fips>
   <need-client-auth>false</need-client-auth>
</ssl>
<http-pubsub>
   <name>pubsub</name>
   <path>/pubsub</path>
   <pub-sub-bean>
      <server-config>
         <name>/pubsub</name>
         <supported-transport>
            <types>
               <element>long-polling</element>
            </types>
         </supported-transport>
         <publish-without-connect-allowed>true</publish-without-connect-allowed>
      </server-config>
      <channels>
         <element>
            <channel-pattern>/evsmonitor</channel-pattern>
         </element>
         <element>
            <channel-pattern>/evsalert</channel-pattern>
         </element>
         <element>
            <channel-pattern>/evsdomainchange</channel-pattern>
         </element>
      </channels>
   </pub-sub-bean>
</http-pubsub>
<cluster>
   <server-name>productionServer</server-name>
</cluster>
<domain>
```

```
    <name>ocep_domain</name>
</domain>
```

Configuration History Management

When you deploy an application to the Oracle Event Processing server, the Oracle Event Processing server creates a configuration history for the application. Any configuration changes you make to the application are recorded in this history. You can view and roll-back (undo) these changes using the Oracle Event Processing Visualizer or `wlevs.Admin` tool.

For more information, see:

- "Configuration History Management" in the *Oracle Fusion Middleware Visualizer User's Guide for Oracle Event Processing*
- Appendix , "Commands for Managing Configuration History"

Configuring the Oracle Event Processing Server bootclasspath

In general, you configure all Oracle Event Processing server options using the Oracle Event Processing server configuration file, you configure all Oracle Event Processing application options using Oracle Event Processing assembly and component configuration files, and you satisfy all Oracle Event Processing application run-time dependencies by importing the relevant private artifacts into an Oracle Event Processing project or deploying a shared application library that contains the relevant shared artifacts that applications import indirectly.

If necessary, you can configure the Oracle Event Processing `bootclasspath` with artifacts, such as native code libraries, that the Oracle Event Processing server makes available before both application libraries and Oracle Event Processing applications are deployed.

You can use this technique to satisfy application or application library dependencies that you cannot satisfy using simple application imports or application libraries.

For more information, see "Application Dependencies" in the *Oracle Fusion Middleware Developer's Guide for Oracle Event Processing for Eclipse*.

How to Configure the Oracle Event Processing Server bootclasspath

Optionally, you can use the `bootclasspath` to make native code libraries available to application libraries that depend on them.

To configure the Oracle Event Processing bootclasspath:

1. Update the server start script in the server directory of your domain directory so that Oracle Event Processing finds the appropriate native library JAR file when it boots up.

   The name of the server start script is `startwlevs.cmd` (Windows) or `startwlevs.sh` (UNIX), and the script is located in the server directory of your domain directory. The out-of-the-box sample domains are located in *`ORACLE_CEP_HOME`*`/ocep_11.1/samples/domains`, and the user domains are located in *`ORACLE_CEP_HOME`*`/user_projects/domains`, where *`ORACLE_CEP_HOME`* refers to the Oracle Event Processing installation directory, such as `d:\oracle_cep`.

   Update the start script by adding the `-Xbootclasspath/a` option to the Java command that executes the `wlevs_3.0.jar` file. Set the `-Xbootclasspath/a` option to the full path name of the native library you are going to use.

For example, if you want to use the native library `mynativelib` located in Oracle Event Processing server directory `%USER_INSTALL_DIR%\bin`, update the `java` command in the start script as follows -- the updated section shown in bold (the example is broken here for readability; in practice you should have the full command on one line):

```
%JAVA_HOME%\bin\java -Dwlevs.home=%USER_INSTALL_DIR% -Dbea.home=%BEA_HOME%
-Xbootclasspath/a:%USER_INSTALL_DIR%\bin\mynativelib.jar
-jar "%USER_INSTALL_DIR%\bin\wlevs_3.0.jar"
-disablesecurity %1 %2 %3 %4 %5 %6
```

In the example, `%USER_INSTALL_DIR%` points to *`ORACLE_CEP_HOME`*`\ocep_11.1`.

2. If Oracle Event Processing is running, restart it so it reads the new `java` option and data source information.

   For more information, see Section , "Starting and Stopping Oracle Event Processing Servers".

Understanding Oracle Event Processing Server Administration Tools

This section describes the various Oracle Event Processing server administration tools that you can use to administer Oracle Event Processing servers, domains, and applications.

You can administer Oracle Event Processing servers, domains, and applications using any of the following:

- Section , "Configuration Wizard"
- Section , "Oracle Event Processing Visualizer"
- Section , "wlevs.Admin Command-Line Utility"
- Section , "Deployer Command-Line Utility"
- Section , "Security Command-Line Utilities"
- Section , "JMX"

For more information, see:

- Section , "Understanding Oracle Event Processing Server Configuration"
- Section , "Understanding Oracle Event Processing Server Administration Tasks"

Configuration Wizard

The Configuration Wizard is a Java application that you can invoke graphically to create and update Oracle Event Processing servers and domains.

For more information, see:

- Chapter 3, "Administering Oracle Event Processing Standalone-Server Domains"
- Chapter 5, "Administering Multi-Server Domains With Oracle Coherence"
- Chapter 7, "Administering Multi-Server Domains With Oracle Event Processing Native Clustering"

Oracle Event Processing Visualizer

The Oracle Event Processing Visualizer is the Oracle Event Processing graphical administration console.

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

Using Oracle Event Processing Visualizer, you can perform a wide variety of Oracle Event Processing server, domain, and application administration tasks for existing Oracle Event Processing standalone server domains and Oracle Event Processing multi-server domains.

For more information, see "Overview of Visualizer" in the *Oracle Fusion Middleware Visualizer User's Guide for Oracle Event Processing*.

wlevs.Admin Command-Line Utility

The `wlevs.Admin` command-line utility is a Java application that you can invoke locally or remotely to perform a wide variety of Oracle Event Processing server, domain, and application administration tasks.

For more information, see Appendix A, "wlevs.Admin Command-Line Reference".

Deployer Command-Line Utility

The Deployer command-line utility is a Java application that you can invoke locally or remotely to perform application deployment and application administration tasks.

For more information, see Appendix B, "Deployer Command-Line Reference".

Security Command-Line Utilities

Oracle Event Processing provides a variety of command-line utilities that simplify security administration.

For more information, see Appendix C, "Security Utilities Command-Line Reference".

JMX

Using standards-based interfaces that are fully compliant with the Java Management Extensions (JMX) specification, you can perform a wide variety of Oracle Event Processing server, domain, and application administration tasks programmatically using JMX and Oracle Event Processing MBeans.

For more information, see:

- Chapter 12, "Configuring JMX for Oracle Event Processing"
- *Oracle Fusion Middleware Java API Reference for Oracle Event Processing*

Understanding Oracle Event Processing Server Administration Tasks

This section describes some of the important Oracle Event Processing server administration tasks, including:

- Section , "Creating Oracle Event Processing Servers and Domains"
- Section , "Updating Oracle Event Processing Servers and Domains"
- Section , "Configuring Oracle Event Processing Servers"
- Section , "Starting and Stopping Oracle Event Processing Servers"
- Section , "Deploying Applications to Oracle Event Processing Servers"

- Section , "Managing Oracle Event Processing Applications, Servers, and Domains"

Creating Oracle Event Processing Servers and Domains

The most important administration task is to create Oracle Event Processing servers and domains.

For more information, see:

- Chapter , "Creating an Oracle Event Processing Standalone-Server Domain"
- Chapter , "Creating an Oracle Event Processing Multi-Server Domain Using Oracle Coherence"
- Chapter , "Creating an Oracle Event Processing Multi-Server Domain Using Oracle Event Processing Native Clustering"

Updating Oracle Event Processing Servers and Domains

Once you create an Oracle Event Processing server and domain, you can update it to change its configuration or group membership.

For more information, see:

- Chapter , "Updating an Oracle Event Processing Standalone-Server Domain"
- Chapter , "Updating an Oracle Event Processing Multi-Server Domain Using Oracle Coherence"
- Chapter , "Updating an Oracle Event Processing Multi-Server Domain Using Oracle Event Processing Native Clustering"

Configuring Oracle Event Processing Servers

Once you create an Oracle Event Processing server and domain, you must configure the various services they provide.

For more information, see:

- Chapter 9, "Configuring Network I/O for Oracle Event Processing"
- Chapter 10, "Configuring Security for Oracle Event Processing"
- Chapter 11, "Configuring Jetty for Oracle Event Processing"
- Section , "Configuring JMX"
- Chapter 13, "Configuring JDBC for Oracle Event Processing"
- Chapter 14, "Configuring HTTP Publish-Subscribe for Oracle Event Processing"
- Chapter 15, "Configuring Logging and Debugging for Oracle Event Processing"

Starting and Stopping Oracle Event Processing Servers

After you have created an Oracle Event Processing domain along with at least a single server, you start a server instance so you can then deploy applications and begin running them. During upgrades and after some configuration changes, you must stop and start the Oracle Event Processing server.

For more information, see:

- Section , "Starting and Stopping an Oracle Event Processing Server in a Standalone-Server Domain"

- Oracle Coherence: Section , "Starting and Stopping an Oracle Event Processing Server in a Multi-Server Domain"
- Oracle Event Processing Native Clustering: Section , "Starting and Stopping an Oracle Event Processing Server in a Multi-Server Domain"

> **Note:** on Windows, do not stop the Oracle Event Processing server by clicking the **Close** button in the command prompt in which you started it. Always stop the Oracle Event Processing server using the `stopwlevs.cmd` script or `Ctrl-C`.

Deploying Applications to Oracle Event Processing Servers

Once you have created and configured an Oracle Event Processing server and domain, you can deploy Oracle Event Processing applications to them.

For more information, see:

- Section , "Deploying an Application to an Oracle Event Processing Standalone-Server Domain"
- Section , "Overview of Deploying an Application to an Oracle Event Processing Multi-Server Domain"
- "Overview of Creating Oracle Event Processing Applications" in the *Oracle Fusion Middleware Developer's Guide for Oracle Event Processing for Eclipse*.

Managing Oracle Event Processing Applications, Servers, and Domains

Once you have deployed applications to an Oracle Event Processing server and domain, you must manage the application to perform tasks such as monitor its performance and perform upgrades.

For more information, see:

- Section , "Understanding Oracle Event Processing Server Administration Tools"
- Section , "Managing With JMX"

Part II

Standalone-Server Domains

Part II contains the following chapters:

- Chapter 2, "Introduction to Standalone-Server Domains"
- Chapter 3, "Administering Oracle Event Processing Standalone-Server Domains"
- Chapter 4, "Deploying Applications to Standalone-Server Domains"

2

Introduction to Standalone-Server Domains

This chapter introduces standalone domains for Oracle Event Processing, or domains that contain a single instance that can be the starting point for clusters.

This chapter includes the following sections:

- Overview of Oracle Event Processing Standalone-Server Domain Administration
- Scalability and Oracle Event Processing Standalone-Server Domain
- Next Steps

Overview of Oracle Event Processing Standalone-Server Domain Administration

An Oracle Event Processing standalone-server domain is a domain that contains a single Oracle Event Processing server. This is the type of domain created by default by the Configuration Wizard and is the starting point for a multi-server domain.

For more information, see:

- Section , "Understanding Oracle Event Processing Servers and Domains"
- Section , "Understanding Oracle Event Processing Server Configuration"

Scalability and Oracle Event Processing Standalone-Server Domain

Using a standalone-server domain, you can take advantage of some of the Oracle Event Processing scalability quality of service options.

To maximize scalability, consider a multi-server domain.

For more information, see:

- "Developing Scalable Applications" in the *Oracle Fusion Middleware Developer's Guide for Oracle Event Processing for Eclipse*
- Chapter 6, "Introduction to Multi-Server Domains"

Next Steps

After creating your own Oracle Event Processing standalone-server domain, consider the administration tasks that Section , "Understanding Oracle Event Processing Server Administration Tasks" describes.

For example, you can:

- Create additional servers in the domain and configure the domain to be multi-server.

  See Chapter 6, "Introduction to Multi-Server Domains."

- Optionally configure the server.

  See Section , "Understanding Oracle Event Processing Server Configuration."

- Create an Oracle Event Processing application.

  See *Oracle Fusion Middleware Developer's Guide for Oracle Event Processing for Eclipse* for a description of the programming model, details about the various components that make up an application, how they all fit together, and typical steps to create a new application.

- Deploy your new, or existing, Oracle Event Processing application to the domain.

  For more information, see:

  - Section , "Deploying an Application to an Oracle Event Processing Standalone-Server Domain"
  - "Assembling and Deploying Oracle Event Processing Applications" in the *Oracle Fusion Middleware Developer's Guide for Oracle Event Processing for Eclipse*

- Manage your applications, servers, and domains:
  - Using the Oracle Event Processing Visualizer.

    See *Oracle Fusion Middleware Visualizer User's Guide for Oracle Event Processing*.
  - Using the `wlevs.Admin` command line tool.

    See Appendix A, "wlevs.Admin Command-Line Reference".
  - Using JMX and MBeans.

    See Section , "Managing With JMX".

3

Administering Oracle Event Processing Standalone-Server Domains

This chapter describes how create and update an Oracle Event Processing standalone-server domain with the configuration wizard, as well as how to start and stop the server.

This chapter includes the following sections:

Creating an Oracle Event Processing Standalone-Server Domain

After you install Oracle Event Processing, use the Configuration Wizard to create a new domain to deploy your applications. The Configuration Wizard creates, by default, the domains in the *ORACLE_CEP_HOME*`/user_projects/domains` directory, where *ORACLE_CEP_HOME* refers to the Oracle Event Processing installation directory such as `d:/oracle_cep`. You can, however, create a domain in any directory you want.

The Configuration Wizard creates a single default server in the domain; all the server-related file are located in a subdirectory of the domain directory named the same as the server (such a `c:\oracle_cep\user_projects\domains\my_domain\defaultserver`). Additionally, the Configuration Wizard allows you to:

- Configure the server's administration user and password.
- Configure the default server to use a database or database driver that is different from the default. In this case, you need to customize the JDBC settings to point to the appropriate database.
- Configure the server's listen port.
- Configure the password for the identity keystore and private keystore.

You can use the Configuration Wizard in the following modes:

- **Graphical mode**—Graphical-mode configuration is an interactive, GUI-based method for creating and configuring a domain. It can be run on both Windows and UNIX systems. See Section , "Creating an Oracle Event Processing Standalone-Server Domain Using the Configuration Wizard in Graphical Mode."

Creating an Oracle Event Processing Standalone-Server Domain Using the Configuration Wizard in Graphical Mode

The following procedure shows how to invoke and use the Configuration Wizard in graphical mode by executing the relevant command script for both Windows or Unix.

Note: On Windows, you can also invoke the Configuration Wizard using the Start menu:

```
Start > All Programs > Oracle Event Processing 11gR1 > Tools >
Configuration Wizard
```

Note: If you intend to add servers from this new domain to the Eclipse IDE, be sure to see "How to Create a Local Oracle Event Processing Server and Server Runtime" in *Oracle Fusion Middleware Developer's Guide for Oracle Event Processing for Eclipse* for information on completing domain creation.

To create an Oracle Event Processing standalone-server domain using the Configuration Wizard in graphical mode:

1. Open a command window and set your environment as described in "Setting Your Development Environment" in the *Oracle Fusion Middleware Developer's Guide for Oracle Event Processing for Eclipse*.
2. Change to the *ORACLE_CEP_HOME*`/ocep_11.1/common/bin` directory, where *ORACLE_CEP_HOME* refers to the main Oracle Event Processing installation directory, such as `/oracle_cep`:

   ```
   prompt> cd /oracle_cep/ocep_11.1/common/bin
   ```

3. Invoke the `config.cmd` (Windows) or `config.sh` (UNIX) command to invoke the wizard:

   ```
   prompt> config.sh
   ```

   After the Configuration Wizard has finished loading, you will see a standard Oracle Welcome window.

   Note: The Oracle Event Processing Configuration Wizard is self-explanatory; however, if you want more information about using the tool, continue reading this procedure.

   ---

4. Click **Next**.
5. In the Choose Create or Update Domain window, choose **Create a New Oracle Event Processing Domain**.
6. Click **Next**.
7. Enter the name of the administrator user for the default server of the domain.
8. Click **Next**.
9. Enter basic configuration information about the default server in the domain. In particular:

- Enter the name of the default server. This name will also be used as the name of the directory that contains the default server files.
- The listen port for Oracle Event Processing itself. Default is 9002.
- The secure listen port. Default is 9003.

10. Click **Next**.
11. Enter and confirm the password for the Oracle Event Processing domain identity keystore.

    By default, the password for the certificate private key will be the same as the password for the identity keystore.

    Note: The Oracle Event Processing Server will not start unless the password for certificate private key is the same as the password for the identity keystore. Do not clear **Use Keystore Password** and do not enter a private key password.

12. Click **Next**.
13. Decide whether or not to update the JDBC datasource configuration:

    The Configuration Wizard bases the creation of a new domain on the Oracle Event Processing domain template; by default, this template does not configure any JDBC datasource for a domain. This means that, unless you change the default domain template used by the Configuration Wizard, if you choose **No** at this step, *no* JDBC data source is configured. If you want to configure a JDBC data source, choose **Yes** at this step to proceed to the page in which you can enter the data source information.

    a. To create a JDBC datasource:
       - Select **Yes**.
       - Click **Next**.
       - Enter the new JDBC datasource values in Configure Database Properties window.

         In the top section, enter the name of the datasource. Then select the database type (Oracle or Microsoft SQL Server) and corresponding drivers; you can also browse to new drivers using the Browse/Append button.

         In the lower section, enter the details about the database to which this data source connects, such as its name, the name of the computer that hosts the database server, the port, and the name and password of the user that connects to the database. The JDBC connection URL is automatically generated for you based on this information.

       - Click **Next**.

    b. To not create a JDBC datasource:
       - Select **No**.
       - Click **Next**.

14. In the Configure Server window, enter the name of the new domain and the full pathname of its domain location.

The configuration wizard creates the domain using its domain name in the domain location directory.

Note: Oracle recommends you always use the default domain location to create your domains:

- On UNIX, the default domain is: *ORACLE_CEP_HOME*/user_projects/domains.
- On Windows, the default domain is: *ORACLE_CEP_HOME*\user_projects\domains.

---

15. Click **Create**.

    If the creation of the domain succeeded, you will see a message similar to the following in the Creating Domain window:

    ```
    Domain created successfully!
    Domain location: C:\oracle_cep\user_projects\domains\ocep_domain
    ```

16. Click **Done**.

    If you intend to add servers from this new domain to the Eclipse IDE, be sure to see "How to Create a Local Oracle Event Processing Server and Server Runtime" in *Oracle Fusion Middleware Developer's Guide for Oracle Event Processing for Eclipse* for information on completing domain creation.

Updating an Oracle Event Processing Standalone-Server Domain

Use the Configuration Wizard to update an existing server in a domain.

The Configuration Wizard creates, by default, the domains in the *ORACLE_CEP_HOME*/user_projects/domains directory, where *ORACLE_CEP_HOME* refers to the Oracle Event Processing installation directory such as d:/oracle_cep. You can, however, create a domain in any directory you want.

The Configuration Wizard creates a single default server in the domain; all the server-related file are located in a subdirectory of the domain directory named the same as the server (such as c:\oracle_cep\user_projects\domains\my_domain\defaultserver).

You can update only the following configuration options of an existing standalone-server in your domain:

- The listen port.
- The configuration of the JDBC datasource.

You can use the Configuration Wizard in the following modes:

- **Graphical mode**—Graphical-mode is an interactive, GUI-based method for updating a domain. It can be run on both Windows and UNIX systems. See Section , "How to Update an Oracle Event Processing Standalone-Server Domain Using the Configuration Wizard in Graphical Mode."

How to Update an Oracle Event Processing Standalone-Server Domain Using the Configuration Wizard in Graphical Mode

The following procedure shows how to invoke and use the Configuration Wizard in graphical mode by executing the relevant command script for both Windows or Unix.

> **Note:** On Windows, you can also invoke the Configuration Wizard using the Start menu:
>
> `Start > All Programs > Oracle Event Processing 11gR1 > Tools > Configuration Wizard`

For clarity, it is assumed in this section that you want to update a server called `productionServer` whose server-related files are located in the `C:\oracle_cep\user_projects\domains\mydomain\productionServer` directory.

To update an Oracle Event Processing standalone-server domain using the Configuration Wizard in graphical mode:

1. Open a command window and set your environment as described in "Setting Your Development Environment" in the *Oracle Fusion Middleware Developer's Guide for Oracle Event Processing for Eclipse*.
2. Change to the *ORACLE_CEP_HOME*`/ocep_11.1/common/bin` directory, where *ORACLE_CEP_HOME* refers to the main Oracle Event Processing installation directory, such as `/oracle_cep`:

   ```
   prompt> cd /oracle_cep/ocep_11.1/common/bin
   ```

3. Invoke the `config.cmd` (Windows) or `config.sh` (UNIX) command to invoke the wizard:

   ```
   prompt> config.sh
   ```

   After the Configuration Wizard has finished loading, you will see a standard Oracle Welcome window.

   > **Note:** The Oracle Event Processing Configuration Wizard is self-explanatory; however, if you want more information about using the tool, continue reading this procedure.

4. Click **Next**.
5. In the Choose Create or Update Domain window, select **Update an existing Oracle Event Processing domain**.
6. Click **Next**.
7. In the text box, enter the full pathname of the server directory that contains the files for the server you want to update.

   In this example, the value is `C:\oracle_cep\user_projects\domains\mydomain\productionServer`.
8. Click **Next**.
9. Update the listen ports for the server.

   > **Note:** To prevent any conflicts when all servers are running at the same time, be sure that you do not enter the same values used by other servers in the domain.

10. Click **Next**.

11. Decide whether or not to update the JDBC datasource configuration:
    a. To update the JDBC datasource configuration:
        – Select **Yes**.
        – Click **Next**.
        – Enter the new JDBC datasource values.
    b. To leave the JDBC datasource configuration unchanged:
        – Select **No**.
        – Click **Next**.
12. Click **Update** to update the server.

Starting and Stopping an Oracle Event Processing Server in a Standalone-Server Domain

You can start and stop an Oracle Event Processing standalone-server using any of the Oracle Event Processing Visualizer, Oracle Event Processing IDE for Eclipse, or command line scripts.

This section describes:

- Section , "How to Start an Oracle Event Processing Standalone-Server Using the startwlevs Script"
- Section , "How to Stop an Oracle Event Processing Standalone-Server Using the stopwlevs Script"

How to Start an Oracle Event Processing Standalone-Server Using the startwlevs Script

Each Oracle Event Processing server directory contains a command script that starts a server instance; by default, the script is called `startwlevs.cmd` (Windows) or `startwlevs.sh` (UNIX).

To start an Oracle Event Processing standalone-server using the startwlevs script:

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

   The server start script is located in the server directory under the main domain directory. For example, the default server directory of the HelloWorld domain is located in *`ORACLE_CEP_HOME`*`/ocep_11.1/samples/domains/helloworld_domain/defaultserver`, where *`ORACLE_CEP_HOME`* refers to the main Oracle Event Processing installation directory, such as `/oracle_cep`.

   a. If using the Oracle JRockit JDK installed with Oracle Event Processing, the `JAVA_HOME` variable should be set as follows:

      For UNIX:

      ```
      JAVA_HOME=ORACLE_CEP_HOME/jrockit_160_20
      ```

      For Windows:

      ```
      set JAVA_HOME=ORACLE_CEP_HOME\jrockit_160_20
      ```

where *ORACLE_CEP_HOME* refers to the installation directory of Oracle Event Processing 10.3, such as `/oracle_cep` (UNIX) or `c:\oracle_cep` (Windows).

b. If using the Oracle JRockit JDK installed with Oracle JRockit Real Time, the `JAVA_HOME` variable should be set as follows:

For UNIX:

```
JAVA_HOME=ORACLE_RT_HOME/JROCKIT_RT_HOME
```

For Windows:

```
set JAVA_HOME=ORACLE_RT_HOME\JROCKIT_RT_HOME
```

where *ORACLE_RT_HOME* refers to the installation directory of Oracle JRockit Real Time, such as `/jrockit` (UNIX) or `c:\jrockit` (Windows), and `JROCKIT_RT_HOME` refers to JRockit Real Time directory.

2. Open a command window and change to the server directory of the domain directory. For example, to start the HelloWorld sample server:

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

3. Execute the `startwlevs.cmd` (Windows) or `startwlevs.sh` (UNIX) script:

```
prompt> startwlevs.cmd
```

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

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

Note: On HP-UX, to avoid an OutOfMemoryError, you may need to increase the MaxPermSize to 256 in startwlevs.sh. For example: -XX:MaxPermSize=256m.

How to Stop an Oracle Event Processing Standalone-Server Using the stopwlevs Script

Each Oracle Event Processing server directory contains a command script that stops a server instance; by default, the script is called `stopwlevs.cmd` (Windows) or `stopwlevs.sh` (UNIX).

Note: The following procedure does not stop an Oracle Event Processing stand-alone server running in SSL mode. To stop an Oracle Event Processing stand-alone server running in SSL mode, run the wlevs.Admin utility, as described in Section , "Running wlevs.Admin Utility in SSL Mode."

To stop an Oracle Event Processing standalone-server using the stopwlevs script:

1. Open a command window and change to the server directory. For example, to stop the running HelloWorld sample server:

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

2. Execute the `stopwlevs.cmd` (Windows) or `stopwlevs.sh` (UNIX) script.

   Use the `-url` argument to pass the URL that establishes a JMX connection to the server you want to stop. This URL takes the form `service:jmx:msarmi://`*`host`*`:`*`port`*`//jndi/jmxconnector`, where *`host`* refers to the computer hosting the server and *`port`* refers to the server's JNDI port, configured in `config.xml` file. For example:

   ```
   prompt> stopwlevs.sh -url service:jmx:msarmi://ariel:9002/jndi/jmxconnector
   ```

   In the example, the host is `ariel` and the JMX port is `9002`. The 9002 port is the `netio` port defined in the Oracle Event Processing server config.xml configuration file. MSA security uses it for JMX connectivity.

   See Section , "Connection Arguments" for additional details about the `-url` argument.

   Note: on Windows, do not stop the Oracle Event Processing server by clicking the **Close** button in the command prompt in which you started it. Always stop the Oracle Event Processing server using the `stopwlevs.cmd` script or `Ctrl-C`.

4

Deploying Applications to Standalone-Server Domains

This chapter describes how to deploy an application to a standalone-server Oracle Event Processing domain, including how to do so with Oracle Event Processing Visualizer and with the deployer utility.

This chapter includes the following section:

- Deploying an Application to an Oracle Event Processing Standalone-Server Domain

Deploying an Application to an Oracle Event Processing Standalone-Server Domain

When you deploy an application to a standalone-server domain, you typically deploy to the singleton server group using either the Oracle Event Processing Visualizer or Deployer utility.

For more information, see:

- Section , "Groups"
- *Oracle Fusion Middleware Visualizer User's Guide for Oracle Event Processing*
- Appendix B, "Deployer Command-Line Reference"

How to Deploy an Application to an Oracle Event Processing Standalone-Server Using the Oracle Event Processing Visualizer

The simplest way to deploy an Oracle Event Processing application to a standalone-server domain is to use the Oracle Event Processing Visualizer.

For more information, see "Deploying an Application" in the *Oracle Fusion Middleware Visualizer User's Guide for Oracle Event Processing*.

How to Deploy an Application to an Oracle Event Processing Singleton Server Group Using the Deployer Utility

If you do not specify a group when you deploy an application, Oracle Event Processing deploys the application to the singleton server group that includes only the specific server to which you deploy the application. This is the standard case in single-server domains.

The following example shows how to deploy to a singleton group; note that the command does not specify a `-group` option:

```
prompt> java -jar wlevsdeploy.jar -url http://ariel:9002/wlevsdeployer -install
myapp_1.0.jar
```

In the example, the `myapp_1.0.jar` application will be deployed to the singleton server group that contains a single server: the one running on host `ariel` and listening to port `9002`.

Part III

Multi-Server Domains

Part III contains the following chapters:

- Chapter 5, "Administering Multi-Server Domains With Oracle Coherence"
- Chapter 6, "Introduction to Multi-Server Domains"
- Chapter 7, "Administering Multi-Server Domains With Oracle Event Processing Native Clustering"
- Chapter 8, "Deploying Applications to Multi-Server Domains"

5

Administering Multi-Server Domains With Oracle Coherence

This chapter describes how to administer Oracle Event Processing multi-server domains that are based on Oracle Coherence. It describes how to create and update multi-server domains, as well as how to secure messages, manage membership changes, and start and stop servers.

This chapter includes the following sections:

Creating an Oracle Event Processing Multi-Server Domain Using Oracle Coherence

This section describes how to create and configure a multi-server domain from two or more Oracle Event Processing servers using Oracle Coherence, including:

Note: To properly configure servers in a multi-server domain, you must configure them with the same multicast address and port and the same domain name. It is an error to configure servers using the same multicast address and port and port but different domain names.

How to Create an Oracle Event Processing Multi-Server Domain With Default Groups Using Oracle Coherence

This procedure describes how to create a multi-server domain that uses only the two predefined groups: the singleton group and domain group. In a domain that uses default groups, all servers must be completely homogenous.

If a domain must support servers that are not completely homogeneous, you configure this by creating custom groups. See Section , "How to Create an Oracle Event Processing Multi-Server Domain With Custom Groups Using Oracle Coherence".

Note: If you are planning to deploy an Oracle Event Processing high availability application, and you require scalability, you may also need to create an Oracle Event Processing high availability notification group. For more information, see "Deployment Group and Notification Group" in the *Oracle Fusion Middleware Developer's Guide for Oracle Event Processing for Eclipse*.

Note: In this section it is assumed that you have already created a domain that contains a single server and that you want to add additional servers to the domain to make it a multi-server domain. See Chapter 3, "Administering Oracle Event Processing Standalone-Server Domains" for details on creating a domain.

To create an Oracle Event Processing multi-server domain with default groups using Coherence:

1. Create a domain that contains a single, initial server.

   See Chapter 3, "Administering Oracle Event Processing Standalone-Server Domains".

2. Add one or more servers to the domain using the Configuration Wizard.

   Note: Even though the Configuration Wizard does not support adding new servers to a multi-server domain, one can use the Configuration Wizard to generate a new stand-alone server, and then manually update its configuration to join a multi-server domain.

   See Section , "Updating an Oracle Event Processing Multi-Server Domain Using Oracle Coherence."

3. Configure all the servers in the multi-server domain by manually editing their `config.xml` files and adding a `cluster` element with specific information.

   To configure the servers in a multi-server domain using default groups, update the `config.xml` file for *each member server* by adding a `cluster` child element of the root `config` element as Example 5–1 shows.

   Include the following child elements of cluster:

   - `server-name`: The `server-name` child element of `cluster` specifies a unique name for the server. Oracle Event Processing Visualizer uses the value of this element when it displays the server in its console.

Default: the Oracle Coherence member name if that is set or `WLEvServer-MEMBERID`.

- `server-host-name`: Specifies the host address/IP used for point-to-point HTTP multi-server communication. Default value is `localhost`.

  This element is mandatory if one or more Oracle Event Processing servers in your multi-server domain are on different hosts and you plan to manage the multi-server domain using the Oracle Event Processing Visualizer. It is also mandatory if a server is deployed on a host machine that has multiple IP addresses configured (whether in a multi-server or standalone-server environment).

- `multicast-address`: The `multicast-address` element is required unless all servers of the multi-server domain are hosted on the same computer; in that case you can omit the `multicast-address` element and Oracle Event Processing automatically assigns a multicast address to the multi-server domain based on the computer's IP address.

  If, however, the servers are hosted on different computers, then you must provide an appropriate domain-local address. Oracle recommends you use an address of the form *`239.255.X.X`*, which is what the auto-assigned multicast address is based on.

  All the Oracle Event Processing servers using this `multicast-address` must be on the same subnet.

  Using Oracle Coherence, you can specify a unicast address here and Oracle Coherence will use WKA (Well Known Addressing).

- `enabled`: By default the clustering of the servers in a multi-server domain is enabled for Oracle Coherence, so the element `<enabled>true</enabled>` is optional.

In Example 5–1, the server is part of a domain called `myDomain`.

Example 5–1 myServer1 config.xml File

```
<config>
  <domain>
        <name>myDomain</name>
  </domain>
  <cluster>
      <server-name>myServer1</server-name>
      <multicast-address>239.255.0.1</multicast-address>
      <enabled>true</enabled>
  </cluster>
...
</config>
```

For each server of the multi-server domain, the `multicast-address` elements must contain the same value. The `server-name` element, however, must be different for each server in the multi-server domain. Example 5–2 shows the `config.xml` file of a second server, called `myServer2`, in the `myDomain` multi-server domain.

Example 5–2 myServer2 config.xml File

```
<config>
  <domain>
        <name>myDomain</name>
  </domain>
```

```
  <cluster>
      <server-name>myServer2</server-name>
      <multicast-address>239.255.0.1</multicast-address>
      <enabled>true</enabled>
  </cluster>
...
</config>
```

See Section , "Order of cluster Element Child Elements" for a description of additional multi-server-related configuration elements and the required order of child elements.

4. Optionally, override the default Oracle Coherence clustering configuration, if necessary.

   See Section , "Configuring the Oracle Coherence Cluster".

5. Optionally, secure the messages that are shared between the servers in a domain by configuring encryption and digital signatures.

   See Section , "How to Secure the Messages Sent Between Servers in a Multi-Server Domain Using Oracle Coherence".

6. To avoid a single point of failure, consider enabling Oracle Event Processing Visualizer on a small subset of `n` machines in the domain.

   See "How to Start Oracle Event Processing Visualizer in a Multi-Server Domain" in the *Oracle Fusion Middleware Visualizer User's Guide for Oracle Event Processing*.

   Note: Enabling Oracle Event Processing Visualizer on a given Oracle Event Processing Server may impact the performance of the server depending on the Oracle Event Processing Visualizer workload.

7. Start all servers in your multi-server domain.

   See Section , "Starting and Stopping an Oracle Event Processing Server in a Multi-Server Domain".

How to Create an Oracle Event Processing Multi-Server Domain With Custom Groups Using Oracle Coherence

This procedure describes how to create a multi-server domain using Oracle Coherence that uses custom groups to accommodate servers which are not completely homogenous.

If all the servers in your domain are completely homogeneous, you do not need to create custom groups. Instead, you can use the predefined, default groups: the singleton group and domain group. See Section , "How to Create an Oracle Event Processing Multi-Server Domain With Default Groups Using Oracle Coherence".

Note: If you are planning to deploy an Oracle Event Processing high availability application, and you require scalability, you may also need to create an Oracle Event Processing high availability notification group. For more information, see "Deployment Group and Notification Group" in the *Oracle Fusion Middleware Developer's Guide for Oracle Event Processing for Eclipse*.

> **Note:** In this section it is assumed that you have already created a domain that contains a single server and that you want to add additional servers to the domain to make it a multi-server domain. See Chapter 3, "Administering Oracle Event Processing Standalone-Server Domains" for details on creating a domain.

In this procedure, assume you have created three servers: `myServer1`, `myServer2`, and `myServer3`. You want `myServer1` to be a member of the `selector` group and `myServer2` and `myServer3` to be members of the `strategy` group.

To create an Oracle Event Processing multi-server domain with custom groups using Oracle Coherence:

1. Create a domain that contains a single, initial server.

   See Chapter 3, "Administering Oracle Event Processing Standalone-Server Domains".

2. Add one or more servers to the domain using the Configuration Wizard.

   > **Note:** Even though the Configuration Wizard does not support adding new servers to a multi-server domain, one can use the Configuration Wizard to generate a new stand-alone server, and then manually update its configuration to join a multi-server domain.

   See Section , "Updating an Oracle Event Processing Multi-Server Domain Using Oracle Coherence."

3. Configure all the servers in the multi-server domain by manually editing their `config.xml` files and adding a `cluster` element with specific information.

   To configure the servers in a multi-server domain using custom groups, update the `config.xml` file for *each member server* by adding (if one does not already exist) a `groups` child element of `cluster` and specifying the name of the group as the value of the `groups` element.

   > **Note:** When adding `cluster` element child elements, observe the correct element order as Section , "Order of cluster Element Child Elements" describes.

   The `groups` element can include more than one group name in the case that the server is a member of more than one group; separate multiple group names using commas.

   The `groups` element is optional; if a server configuration does not include one, then the server is a member of the default groups (domain and singleton). For more information about the domain and singleton groups, see Section , "Groups".

   Example 5–3, Example 5–4, and Example 5–5 show the relevant snippets of the `config.xml` file for each server.

Example 5–3 Server Configuration File config.xml for myServer1

```
<config>
  <domain>
```

```
        <name>myDomain</name>
  </domain>
  <cluster>
      <server-name>myServer1</server-name>
      <multicast-address>239.255.0.1</multicast-address>
      <enabled>true</enabled>
      <groups>selector</groups>
  </cluster>
...
</config>
```

Example 5–4 Server Configuration File config.xml for myServer2

```
<config>
  <domain>
        <name>myDomain</name>
  </domain>
  <cluster>
      <server-name>myServer2</server-name>
      <multicast-address>239.255.0.1</multicast-address>
      <enabled>true</enabled>
      <groups>strategy</groups>
  </cluster>
...
</config>
```

Example 5–5 Server Configuration File config.xml for myServer3

```
<config>
  <domain>
        <name>myDomain</name>
  </domain>
  <cluster>
      <server-name>myServer3</server-name>
      <multicast-address>239.255.0.1</multicast-address>
      <enabled>true</enabled>
      <groups>strategy</groups>
  </cluster>
...
</config>
```

4. Optionally, override the default Oracle Coherence clustering configuration, if necessary.

   See Section , "Configuring the Oracle Coherence Cluster".

5. Optionally, secure the messages that are shared between the servers in a domain by configuring encryption and digital signatures.

   See Section , "How to Secure the Messages Sent Between Servers in a Multi-Server Domain Using Oracle Coherence".

6. To avoid a single point of failure, consider enabling Oracle Event Processing Visualizer on a small subset of `n` machines in the domain.

   See "How to Start Oracle Event Processing Visualizer in a Multi-Server Domain" in the *Oracle Fusion Middleware Visualizer User's Guide for Oracle Event Processing*.

> **Note:** Enabling Oracle Event Processing Visualizer on a given Oracle Event Processing Server may impact the performance of the server depending on the Oracle Event Processing Visualizer workload.

7. Start all servers in your multi-server domain.

   See Section , "Starting and Stopping an Oracle Event Processing Server in a Multi-Server Domain".

Configuring the Oracle Coherence Cluster

Oracle Event Processing leverages the native configuration provided by Oracle Coherence, the `tangosol-coherence-override.xml` for Oracle Coherence cluster configuration. See Section , "The tangosol-coherence-override.xml File" for information about this file as well as an example.

When assembling your application, consider the following:

- `tangosol-coherence-override.xml` is a global per-server file (referred to as "operational configuration" in the Oracle Coherence documentation); put this file in the same place as the server configuration file, in the Oracle Event Processing server `config` directory.

When you declare that a caching system uses the Oracle Coherence provider, be sure that all of the caches of this caching system also map to an Oracle Coherence configuration and not an Oracle Event Processing local configuration, or Oracle Event Processing throws an exception.

The tangosol-coherence-override.xml File

The `tangosol-coherence-override.xml` file configures Oracle Coherence clustering. The following elements are supported for use from the override file: `<cluster-config>`, `<management-config>`, and `<logging-config>`.

> **Note:** You cannot override the cluster name as Oracle Event Processing will always set the cluster name to the domain name. Consider choosing use a unique name for each Oracle Event Processing domain to prevent accidental cluster discovery between different domains.

The following sample shows a simple configuration that specifies the time-to-live setting that determines the maximum number of "hops" a packet may traverse, where a hop is measured as a traversal from one network segment to another via a router.

```
<?xml version='1.0'?>
<coherence xml-override="/tangosol-coherence-override.xml">
  <cluster-config>
    <multicast-listener>
      <time-to-live>3</time-to-live>
    </multicast-listener>
    ...
</coherence>
```

For detailed information about the `tangosol-coherence-override.xml` file, see "Operational Override File (tangosol-coherence-override.xml)" in the *Oracle Coherence Developer's Guide*.

Updating an Oracle Event Processing Multi-Server Domain Using Oracle Coherence

Use the Configuration Wizard to add a new server to an existing standalone server domain so as to later convert it into a multi-server domain. The procedure is similar to creating a new domain, so be sure you read Section , "Creating an Oracle Event Processing Standalone-Server Domain" before continuing with this section.

How to Update an Oracle Event Processing Multi-Server Domain Using the Configuration Wizard in Graphical Mode

The following procedure shows how to invoke and use the Configuration Wizard in graphical mode by executing the relevant command script for both Windows or Unix.

Note: On Windows, you can also invoke the Configuration Wizard using the Start menu:

```
Start > All Programs > Oracle Event Processing 11gR1 > Tools >
Configuration Wizard
```

For clarity, it is assumed that:

- You have already created a new domain and its domain directory is `C:\oracle_cep\user_projects\domains\mydomain`.
- The domain includes a single server called `defaultserver` and the server files are located in the `C:\oracle_cep\user_projects\domains\myDomain\myServer1` directory.
- You want to create a new server in the existing `mydomain` domain called `myServer2`.

To update an Oracle Event Processing multi-server domain using the Configuration Wizard in graphical mode:

1. Open a command window and set your environment as described in "Setting Your Development Environment" in the *Oracle Fusion Middleware Developer's Guide for Oracle Event Processing for Eclipse*.
2. Change to the *`ORACLE_CEP_HOME`*`/ocep_11.1/common/bin` directory, where *`ORACLE_CEP_HOME`* refers to the main Oracle Event Processing installation directory, such as `/oracle_cep`:

   ```
   prompt> cd /oracle_cep/ocep_11.1/common/bin
   ```

3. Invoke the `config.cmd` (Windows) or `config.sh` (UNIX) command to invoke the wizard:

   ```
   prompt> ./config.sh
   ```

   After the Configuration Wizard has finished loading, you will see a standard Oracle Welcome window.

   Note: The Oracle Event Processing Configuration Wizard is self-explanatory; however, if you want more information about using the tool, continue reading this procedure.

   ---

4. Click **Next**.

5. In the Choose Create or Update Domain window, choose **Create a New Oracle Event Processing Domain**.
6. Click **Next**.
7. Enter the name and password of the administrator user for the new server you are adding to the domain.'
8. Click **Next**.
9. Enter basic configuration information about the new server in the domain. *If the new server is located on the same computer as any other servers in the domain, be sure the following information is different from that of the other servers to prevent conflicts when starting all servers.* In particular:
   - Enter the name of the new server. This name will also be used as the name of the directory that contains the new server's files. Following our example, this value is `myServer1`.
   - The listen port for Oracle Event Processing itself.
10. Click **Next**.
11. Enter and confirm the password for the Oracle Event Processing identity keystore. By default, the password for the certificate private key will be the same as the identity keystore; if you want it to be different, clear Use Keystore Password and enter the private key password.
12. Click **Next**.
13. In the Configuration Options window, choose:
    a. **Yes** if you want to change the default JDBC data source configuration,
    b. **No** to accept the defaults.

    Note: When you deploy an application to a group in the domain, Oracle Event Processing replicates the application to each server that is a member of the group. This means that if your application uses a datasource, and you have configured the datasource differently for each server in the domain, then the storage and retrieval of data to and from this data source will differ depending on the server on which the application is running.
14. Click **Next**.
15. If you chose to change the default JDBC data source configuration, enter the information in the Configure Database Properties window.

    In the top section, enter the name of the datasource. Then select the database type (Oracle or Microsoft SQL Server) and corresponding drivers; you can also browse to new drivers using the Browse/Append button.

    In the lower section, enter the details about the database to which this data source connects, such as its name, the name of the computer that hosts the database server, the port, and the name and password of the user that connects to the database. The JDBC connection URL is automatically generated for you based on this information.
16. Click **Next**.

17. In the Configure Server window, enter the name of the *existing* domain and the full pathname of its location. Following our example, you would enter `myDomain` for the domain name and `C:\oracle_cep\user_projects\domains` for the domain location.
18. Click **Create**.
19. If the creation of the new server succeeded, you will see a message similar to the following in the Creating Domain window:

    ```
    Domain created successfully!
    Domain location: /oracle_cep/user_projects/domains/myDomain
    ```

20. Click **Done**.

Securing the Messages Sent Between Servers in a Multi-Server Domain

The servers in a multi-server domain update their state by exchanging multi-server-related messages. It is important that these messages be at least checked for integrity. A private key can be used to achieve integrity. This key must be shared by all servers in the domain.

This section describes:

- Section , "How to Secure the Messages Sent Between Servers in a Multi-Server Domain Using Oracle Coherence"

How to Secure the Messages Sent Between Servers in a Multi-Server Domain Using Oracle Coherence

You can secure the messages sent between servers in a multi-server domain using the Oracle Coherence clustering implementation.

To secure the messages sent between servers in a multi-server domain using Oracle Coherence:

1. Stop all servers in your multi-server domain, if they are currently running.

   See Section , "Starting and Stopping an Oracle Event Processing Server in a Multi-Server Domain".

2. Edit the `config.xml` file of each server in the multi-server domain by adding the `security` child element to the `cluster` element, as Example 5–6 shows.

Example 5–6 The cluster Element security Child Element

```
<config>
  <domain>
        <name>myDomain</name>
  </domain>
  <cluster>
      <server-name>myServer1</server-name>
      <multicast-address>239.255.0.1</multicast-address>
      <identity>1</identity>
      <enabled>coherence</enabled>
      <security>encrypt</security>
  </cluster>
...
</config>
```

The `config.xml` file is located in the *`DOMAIN_DIR`*`/`*`servername`*`/config` directory of each server, where *`DOMAIN_DIR`* refers to the domain directory and *`servername`* refers to the name of your server, such as `d:\oracle_cep_home\user_projects\domains\mydomain\myserver1\config`.

You must specify one of the following values for the `security` child element:

- `none`—Default value. Specifies that no security is configured for the multi-server domain.
- `encrypt`—Specifies that multi-server messages should be encrypted.

Observe the correct order of child elements in the cluster element. See Section , "Order of cluster Element Child Elements".

3. Edit the *`DOMAIN_DIR`*`/servername/config/security-config.xml` file of each server in the multi-server domain by adding the `encryption-service` child element of the `config` root element, as Example 5–7 shows.

Example 5–7 The security-config.xml File encryption-service Element

```
<config>
  <encryption-service>
     <signature-enabled>true</signature-enabled>
  </encryption-service>
  <css-realm>
   ...
</config>
```

4. Ensure that the *`DOMAIN_DIR`*`/`*`servername`*`/.aesinternal.dat` file for each server in the multi-server domain is exactly the same by copying the file from one server to the other servers.

   This file is automatically created by the Configuration Wizard when you first created the server; Oracle Event Processing uses this file for encrypting messages.

   For example, assume all the servers in your domain are located in the `d:\oracle_cep\user_projects\domains\mydomain` directory, and that the domain has three servers: `server1`, `server2`, and `server3`. To ensure they all have the same `.aesinternal.dat` file, copy the one from `server1` to the other servers:

   ```
   prompt> cd d:\oracle_cep\user_projects\domains\mydomain\server1
   prompt> cp .aesinternal.dat ..\server2
   prompt> cp .aesinternal.dat ..\server3
   ```

5. Start one of the servers in your domain.

   See Section , "Starting and Stopping an Oracle Event Processing Server in a Multi-Server Domain".

   Because of the `encryption-service` element that you added to the `security-config.xml` file in step 3, Oracle Event Processing automatically creates the `.msasig.dat` file in the main server directory. Oracle Event Processing uses this file for digitally signing messages.

6. Stop the server you just started.

   See Section , "Starting and Stopping an Oracle Event Processing Server in a Multi-Server Domain".

7. Copy the `.msasig.dat` file you created in step 5 to the other servers.

   For example:

```
prompt> cd d:\oracle_cep\user_projects\domains\mydomain\server1
prompt> cp .msasig.dat ..\server2
prompt> cp .msasig.dat ..\server3
```

8. Perform the following steps on each server in the cluster:
   - Open a command window and set your environment as described in "Setting Your Development Environment" in the *Oracle Fusion Middleware Developer's Guide for Oracle Event Processing for Eclipse*.
   - Change to the *DOMAIN_DIR*/*servername* directory, where *DOMAIN_DIR* refers to the domain directory and *servername* refers to the name of your server, such as `d:\oracle_cep_home\user_projects\domains\mydomain\myserver1`.

     ```
     prompt> cd d:\oracle_cep_home\user_projects\domains\mydomain\myserver1
     ```

   - Create a keystore `coherence-identity.jks` containing the boot user using the JDK `keytool` utility and the following command line (broken here for readability; in practice the full command should be on one line):

     ```
     prompt> keytool -genkey -v -keystore config/coherence-identity.jks
     -storepass PASSWORD -alias BOOT-USER -keypass BOOT-USER-PASSWORD
     -dname CN=BOOT-USER
     ```

     Where:
     - *PASSWORD* is the password used to secure the keystore.
     - *BOOT-USER* is the user name you used to log into the Oracle Event Processing server host.
     - *BOOT-USER-PASSWORD* is the password you used when you logged into the Oracle Event Processing server host.
   - Create a `permissions.xml` file.
   - Edit the `permissions.xml` file to add the following permission for the boot user:

     ```
     <permissions>
         <grant>
             <principal>
                 <class>javax.security.auth.x500.X500Principal</class>
                 <name>CN=BOOT-USER</name>
             </principal>

             <permission>
                 <target>*</target>
                 <action>all</action>
             </permission>
         </grant>
     </permissions>
     ```

     Where *BOOT-USER* is the user name you used to log into the Oracle Event Processing server host.
   - Save and close the `permissions.xml` file.
   - Create a `login.config` file.
   - Edit the `login.config` file to add the following:

     ```
     Coherence {
         com.tangosol.security.KeystoreLogin required
     ```

```
        keyStorePath=".${/}config${/}coherence-identity.jks";
};
```

- Save and close the `login.config` file.
- Update the server startup script for your platform, `startwlevs.cmd` (Windows) or `startwlevs.sh` (UNIX), by adding the following property to the `java` command that actually starts the server:

  ```
  -Djava.security.auth.login.config=./login.config
  ```

  For example (in practice, the full command should be on one line):

  ```
  "%JAVA_HOME%\bin\java" %DGC% %DEBUG%
  -Djava.security.auth.login.config=./login.config
  -Dwlevs.home="%USER_INSTALL_DIR%" -Dbea.hoe="%BEA_HOME%"
  -jar "%USER_INSTALL_DIR%\bin\wlevs.jar" %1 %2 %3 %4 %5 %6
  ```

9. If you plan to use Oracle Event Processing Visualizer with the servers in this domain, see Section , "How to Configure SSL in a Multi-Server Domain for Oracle Event Processing Visualizer".
10. Start all servers in your multi-server domain.

    See Section , "Starting and Stopping an Oracle Event Processing Server in a Multi-Server Domain".

Using the Multi-Server Domain APIs to Manage Group Membership Changes

In an active-active system, applications are deployed homogeneously across several servers and are actively executing.

There are cases, however, when these homogeneously-deployed applications need to elect a primary one as the coordinator or leader. In this case, events that result from the coordinator application are kept and passed on to the next component in the EPN; the results of secondary servers are dropped. However, if the coordinator fails, then one of the secondary servers must be elected as the new coordinator.

To enable this in an application, the adapter or event bean, generally in the role of an event sink, must implement the `com.bea.wlevs.ede.api.cluster.GroupMembershipListener` interface which allows the event sinks to listen for multi-server domain group membership changes. At runtime, Oracle Event Processing automatically invokes the `onMembershipChange` callback method whenever membership changes occur.

The signature of the callback method is as follows:

```
onMembershipChange(Server localIdentity, Configuration groupConfiguration);
```

In the implementation of the `onMembershipChange` callback method, the event sink uses the `Server` object (`localIdentity`) to verify if it is the leader. This can be done be comparing `localIdentity` with the result of `Configuration.getCoordinator()` run on the second parameter, `groupConfiguration`. This parameter also allows a server to know what the current members of the group are by executing `Configuration.getMembers()`.

> **Note:** There is a new API for notification groups. For more information, see "Deployment Group and Notification Group" in the *Oracle Fusion Middleware Developer's Guide for Oracle Event Processing for Eclipse*.

In order to only keep events if it is a coordinator, the event sink must get a new `Server` identity every time membership in the group changes. Group membership changes occur if, for example, another server within the group fails and is no longer the coordinator.

A similar interface `com.bea.wlevs.ede.api.cluster.DomainMembershipListener` exists for listening to membership changes to the domain as a whole, rather than just changes to the group.

Note that in a hot-hot configuration, there is a non-zero delay in failure notification. If you are using the notification APIs to implement clustering, you will lose and not process events that occur in the window between the server failure and the notification being delivered to the new master server.

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

Starting and Stopping an Oracle Event Processing Server in a Multi-Server Domain

To start the servers in a multi-server domain, start each server separately by running its start script. This is the same way you start a server in a standalone server domain. See Section , "Starting and Stopping an Oracle Event Processing Server in a Standalone-Server Domain" for details.

If you have not configured custom groups for the multi-server domain, then all servers are members of just the pre-defined domain group, which contains all the servers in the multi-server domain, and a singleton group, one for each member server. This means, for example, if there are three servers in the multi-server domain then there are three singleton groups.

If, however, you have configured custom groups for the multi-server domain, then the servers are members of the groups for which they have been configured, as well as the pre-defined groups.

> **Note:** on Windows, do not stop the Oracle Event Processing server by clicking the **Close** button in the command prompt in which you started it. Always stop the Oracle Event Processing server using the `stopwlevs.cmd` script or `Ctrl-C`.

6

Introduction to Multi-Server Domains

This chapter introduces Oracle Event Processing multi-server domains, also known as clusters, based on Oracle Coherence or Oracle Event Processing native technology. It also provides overviews of multi-server messaging, high availability, and scalability.

This chapter includes the following sections:

Overview of Oracle Event Processing Multi-Server Domain Administration

An Oracle Event Processing multi-server domain (or cluster) is a domain to which you can add one or more servers that become logically connected for the purposes of management, and physically connected using a shared User Datagram Protocol (UDP) multicast address and port. All servers in an Oracle Event Processing multi-server domain are aware of all other servers in the domain and any one server can be used as an access point for making changes to the deployments in the domain.

Management of the multi-server infrastructure is done at the domain level. Thus server failure, start, or restart is detected by every member of the multi-server domain. Each member of the multi-server domain has a consistent, agreed notion of domain membership enforced by the multi-server infrastructure.

To properly configure servers in a multi-server domain, you must configure them with the same multicast address and port and the same domain name. It is an error to configure servers using the same multicast address and port and port but different domain names.

Applications deployed to the default group in a multi-server domain are deployed homogeneously to all servers of that domain, therefore all servers must have the appropriate configuration resources required by the application.

Oracle Event Processing supports the following clustering systems:

- Section , "Oracle Event Processing Native Clustering"

The rest of this chapter describes:

- Section , "Groups"
- Section , "Multi-Server Notifications and Messaging"
- Section , "Multi-Server Domain Directory Structure"
- Section , "Order of cluster Element Child Elements"
- Section , "High Availability and Multi-Server Domains"
- Section , "Scalability and Multi-Server Domains"
- Section , "Next Steps"

For more information, see:

- Section , "Understanding Oracle Event Processing Servers and Domains"
- Section , "Understanding Oracle Event Processing Server Configuration"

Oracle Coherence Clustering

Oracle Coherence: provides replicated and distributed (partitioned) data management services on top of a reliable, highly scalable peer-to-peer clustering protocol. Oracle Coherence has no single points of failure; it automatically and transparently fails over and redistributes its clustered data management services when a server becomes inoperative or is disconnected from the network. When a new server is added, or when a failed server is restarted, it automatically joins the cluster and Oracle Coherence fails back services to it, transparently redistributing the cluster load.

Note: Before you can use Oracle Event Processing with Oracle Coherence, you must obtain a valid Oracle Coherence license such as a license for Coherence Enterprise Edition, Coherence Grid Edition, or Oracle WebLogic Application Grid. For more information on Oracle Coherence, see `http://www.oracle.com/technology/products/coherence/index.html`.

Using Oracle Coherence, you can take advantage of Oracle Event Processing high availability quality of service options.

For more information, see:

- Section , "Creating an Oracle Event Processing Multi-Server Domain Using Oracle Coherence"
- Section , "High Availability and Multi-Server Domains"

Oracle Event Processing Native Clustering

Oracle Event Processing native clustering: provides a native clustering implementation based on TOTEM.

Using Oracle Event Processing native clustering, you cannot take advantage of Oracle Event Processing high availability quality of service options.

For more information, see Section , "Creating an Oracle Event Processing Multi-Server Domain Using Oracle Event Processing Native Clustering".

Groups

In order to support the deployment to, and management of, the multi-server domain at a finer grained-level than the domain, Oracle Event Processing introduces the concept of *groups*. A group is a set of one or more servers with a unique name within the domain. In an Oracle Event Processing domain, an arbitrary number of groups may exist with a configurable group membership. A server may be a member of more than one group, although typically this information is transparent to the user.

When you deploy an application to a multi-server domain, you deploy it to a particular group. Applications deployed to any group must have a unique name across the domain.

Applications deployed to a group in a multi-server domain are deployed homogeneously to all servers of that group, therefore all servers must have the appropriate configuration resources required by the application.

An application that is deployed to a server can be uninstalled from another server under the same domain.

The following pre-defined deployment groups always exist:

- Section , "Singleton Server Deployment Group"
- Section , "Domain Deployment Group"

Alternatively, you can create a custom deployment group (see Section , "Custom Deployment Groups").

Note: If you are planning to deploy an Oracle Event Processing high availability application, and you require scalability, you may also need to create an Oracle Event Processing high availability notification group. For more information, see "Deployment Group and Notification Group" in the *Oracle Fusion Middleware Developer's Guide for Oracle Event Processing for Eclipse*.

Singleton Server Deployment Group

This group consists of only the local server. This means that the membership of this group depends on the server from which it is accessed. This group can be used to pin deployments to a single server.

For more information, see Section , "How to Create an Oracle Event Processing Multi-Server Domain With Default Groups Using Oracle Coherence".

Domain Deployment Group

This group contains all live members of the domain. Its membership is automatically managed and cannot be changed by the user.

The domain name is determined by the Oracle Event Processing server `config.xml` file `domain` element. For example, the domain is named `mydomain` if your `config.xml` file is like this:

```
<domain>
    <name>mydomain</name>
</domain>
```

The default name is `WLEventServerDomain`.

For more information, see Section , "How to Create an Oracle Event Processing Multi-Server Domain With Default Groups Using Oracle Coherence".

Custom Deployment Groups

There are cases where the application logic cannot simply be replicated across a homogenous set of servers in a multi-server domain. Examples of these types of applications are those that must determine the best price provided by different pricing engines, or applications that send an alert when a position crosses a threshold. In these cases, the application is not idempotent; it must calculate only once or send a single event. In other cases, the application has a singleton nature, such as a monitoring application, the HTTP pub-sub server, and so on.

As a more complex example, consider a domain that has two applications: the `strategies` application uses several strategies for calculating different prices for some derivative and then feeds its results to a `selector` application. The `selector` application then selects the best price amongst the different options provided by the `strategies'` application results. The `strategies` application can be replicated to achieve fault-tolerance. However, the `selector` application must be able to keep state so as to determine the best price; for this reason, the `selector` application *cannot* be replicated in a hot/hot fashion.

If a domain must support servers that are not completely homogeneous, you configure this by creating custom groups.

Applications deployed to a custom group in a multi-server domain are deployed homogeneously to all servers of that group, therefore all servers must have the appropriate configuration resources required by the application.

For more information, see Section , "How to Create an Oracle Event Processing Multi-Server Domain With Custom Groups Using Oracle Coherence".

Multi-Server Notifications and Messaging

In order to provide high availability (HA)-like capabilities to adapter and event bean implementations, Oracle Event Processing provides a number of notification and messaging APIs at both the group- and server-level. Using these APIs, you can configure a server to receive notification when its group or domain membership changes, either because an administrator deliberately changed it or due to a server failure. Similarly you can use these APIs to send messages to both individual groups and to the domain.

When you configure your application to use Oracle Event Processing high availability options, the primary Oracle Event Processing server uses Oracle Coherence to communicate with its secondary servers to keep them up to date with the primary server's event processing progress.

You can configure Oracle Event Processing servers in a multi-server domain to communicate securely.

For more information, see:

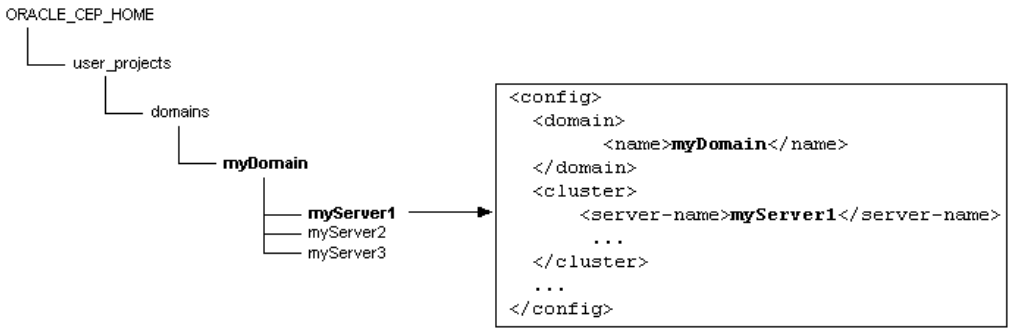
- Section , "How to Secure the Messages Sent Between Servers in a Multi-Server Domain Using Oracle Coherence"
- Section , "How to Secure the Messages Sent Between Servers in a Multi-Server Domain Using Oracle Event Processing Native Clustering"
- "Understanding High Availability" in the *Oracle Fusion Middleware Developer's Guide for Oracle Event Processing for Eclipse*

Multi-Server Domain Directory Structure

Servers in an Oracle Event Processing domain store their files in a single directory. By convention, the directories of the servers in a multi-server domain are sub-directories of the domain directory. Additionally, the name of the servers and domain correspond to the name of the server directories and domain directory, respectively. This is by convention only, and not required, although Oracle recommends you set up your domains this way for simplicity and consistency. If the servers of the multi-server domain are located on different computers, you can replicate the directory structure on both computers, also for simplicity and consistency.

Figure 6–1 shows a multi-server domain directory with three servers.

Figure 6–1 Multi-Server Domain Directory Structure



The `myServer1` configuration file snippet shows how the domain directory and domain object are configured with the same name, as well as the server directory and server name. The domain directory is located in the *ORACLE_CEP_HOME*`/user_projects/domains` directory, which is the default location for Oracle Event Processing domains.

Order of cluster Element Child Elements

The order of `cluster` element child elements in the `config.xml` file is important; if you include elements in the incorrect order you may encounter an error. The following list describes the order in which you should list the child elements:

- `server-name`
- `server-host-name`: Specifies the host address/IP used for point-to-point HTTP multi-server communication. Default value is `localhost`.

  This element is mandatory if one or more Oracle Event Processing servers in your multi-server domain are on different hosts and you plan to manage the multi-server domain using the Oracle Event Processing Visualizer. It is also mandatory if a server is deployed on a host machine that has multiple IP addresses configured (whether in a multi-server or standalone-server environment).
- `multicast-address`: The multicast communication address. For Coherence well-known addressing (WKA) a unicast address can be used.
- `multicast-port`: Optional. Specifies the port used for multicast traffic. Default value is `9001`.
- `identity`: Mandatory only for Oracle Event Processing native clustering; not used for Oracle Coherence.

- `enabled`
- `security`
- `groups`
- `operation-timeout`: Optional. Specifies, in milliseconds, the timeout for point-to-point HTTP multi-server requests. Default value is `30000`.

For a complete description of the Oracle Event Processing server `config.xml` file `cluster` element, see "cluster" in the *Oracle Fusion Middleware Developer's Guide for Oracle Event Processing for Eclipse*.

High Availability and Multi-Server Domains

If you use Oracle Coherence clustering for your multi-server domain, you can take advantage of Oracle Event Processing high availability quality of service options. These options are not supported using Oracle Event Processing native clustering.

For more information, see "Understanding High Availability" in the *Oracle Fusion Middleware Developer's Guide for Oracle Event Processing for Eclipse*.

Scalability and Multi-Server Domains

Using either Oracle Coherence or Oracle Event Processing native clustering, you can take advantage of Oracle Event Processing scalability quality of service options.

For more information, see "Developing Scalable Applications" in the *Oracle Fusion Middleware Developer's Guide for Oracle Event Processing for Eclipse*.

Next Steps

After creating your own Oracle Event Processing multi-server domain, consider the administration tasks that Section , "Understanding Oracle Event Processing Server Administration Tasks" describes.

For example, you can:

- Optionally configure the server.

  See Section , "Understanding Oracle Event Processing Server Configuration."

- Create an Oracle Event Processing application.

  See *Oracle Fusion Middleware Developer's Guide for Oracle Event Processing for Eclipse* for a description of the programming model, details about the various components that make up an application, how they all fit together, and typical steps to create a new application.

- Deploy your new, or existing, Oracle Event Processing application to the domain.

  For more information, see:

  - Section , "Overview of Deploying an Application to an Oracle Event Processing Multi-Server Domain"
  - "Assembling and Deploying Oracle Event Processing Applications" in the *Oracle Fusion Middleware Developer's Guide for Oracle Event Processing for Eclipse*

- Manage your applications, servers, and domains:
  - Using the Oracle Event Processing Visualizer.

See *Oracle Fusion Middleware Visualizer User's Guide for Oracle Event Processing*.

- Using the `wlevs.Admin` command line tool.

  See Appendix A, "wlevs.Admin Command-Line Reference".

- Using JMX and MBeans.

  See Section , "Managing With JMX".

7

Administering Multi-Server Domains With Oracle Event Processing Native Clustering

This chapter describes how to administer Oracle Event Processing multi-server domains that are based on Oracle Event Processing native clustering. It describes how to create and update these domains, secure messages within them, manage group members, and start and stop servers.

This chapter includes the following sections:

- Creating an Oracle Event Processing Multi-Server Domain Using Oracle Event Processing Native Clustering
- Updating an Oracle Event Processing Multi-Server Domain Using Oracle Event Processing Native Clustering
- Securing the Messages Sent Between Servers in a Multi-Server Domain
- Using the Multi-Server Domain APIs to Manage Group Membership Changes
- Starting and Stopping an Oracle Event Processing Server in a Multi-Server Domain

Creating an Oracle Event Processing Multi-Server Domain Using Oracle Event Processing Native Clustering

This section describes how to create and configure a multi-server domain from two or more Oracle Event Processing servers using Oracle Event Processing native clustering. This section includes the following:

- Section , "How to Create an Oracle Event Processing Multi-Server Domain With Default Groups Using Oracle Event Processing Native Clustering,"
- Section , "How to Create an Oracle Event Processing Multi-Server Domain With Custom Groups Using Oracle Event Processing Native Clustering,"

> **Note:** To properly configure servers in a multi-server domain, you must configure them with the same multicast address and port and the same domain name. It is an error to configure servers using the same multicast address and port and port but different domain names.

How to Create an Oracle Event Processing Multi-Server Domain With Default Groups Using Oracle Event Processing Native Clustering

This procedure describes how to create a multi-server domain that uses only the two predefined groups: the singleton group and domain group. In a domain that uses default groups, all servers must be completely homogenous.

If a domain must support servers that are not completely homogeneous, you configure this by creating custom groups. See Section , "How to Create an Oracle Event Processing Multi-Server Domain With Custom Groups Using Oracle Event Processing Native Clustering".

For more information about default and custom groups, see Section , "Groups".

Note: In this section it is assumed that you have already created a domain that contains a single server and that you want to add additional servers to the domain to make it a multi-server domain. See Chapter 3, "Administering Oracle Event Processing Standalone-Server Domains" for details on creating a domain.

To create an Oracle Event Processing multi-server domain with default groups using Oracle Event Processing Native Clustering:

1. Create a domain that contains a single, initial server.

   See Chapter 3, "Administering Oracle Event Processing Standalone-Server Domains".

2. Add one or more servers to the domain using the Configuration Wizard.

   Note: Even though the Configuration WIzard does not support adding new servers to a multi-server domain, one can use the Configuration Wizard to generate a new stand-alone server, and then manually update its configuration to join a multi-server domain.

   See Section , "Updating an Oracle Event Processing Multi-Server Domain Using Oracle Event Processing Native Clustering."

3. Configure all the servers in the multi-server domain by manually editing their `config.xml` files and adding a `cluster` element with specific information.

   To configure the servers in a multi-server domain using default groups, update the `config.xml` file for *each member server* by adding a `cluster` child element of the root `config` element as Example 7–1 shows.

   Include the following child elements of cluster:

   - `server-name`: The `server-name` child element of `cluster` specifies a unique name for the server. Oracle Event Processing Visualizer uses the value of this element when it displays the server in its console. The default value if the element is not set is `Server-`*`identity`* where *`identity`* is the value of the `identity` element.
   - `server-host-name`: Specifies the host address/IP used for point-to-point HTTP multi-server communication. Default value is `localhost`.

     This element is mandatory if one or more Oracle Event Processing servers in your multi-server domain are on different hosts and you plan to manage the multi-server domain using the Oracle Event Processing Visualizer. It is also

mandatory if a server is deployed on a host machine that has multiple IP addresses configured (whether in a multi-server or standalone-server environment).

- `multicast-address`: The `multicast-address` element is required unless all servers of the multi-server domain are hosted on the same computer; in that case you can omit the `multicast-address` element and Oracle Event Processing automatically assigns a multicast address to the multi-server domain based on the computer's IP address.

  If, however, the servers are hosted on different computers, then you must provide an appropriate domain-local address. Oracle recommends you use an address of the form *`239.255.X.X`*, which is what the auto-assigned multicast address is based on.

  All the Oracle Event Processing servers using this `multicast-address` must be on the same subnet.

- `identity`: The `identity` element identifies the server's identity and must be an integer between 1 and `INT_MAX`. Oracle Event Processing numerically compares the server identities during multi-server operations; the server with the lowest identity becomes the domain coordinator. Be sure that each server in the multi-server domain has a different identity; if servers have the same identity, the results of multi-server operations are unpredictable.

- `enabled`: By default the clustering of the servers in a multi-server domain is enabled for Oracle Coherence, so to enable Oracle Event Processing native clustering use `<enabled>evs4j</enabled>`.

Note: When adding `cluster` element child elements, observe the correct element order as Section , "Order of cluster Element Child Elements" describes.

Example 7–1 myServer1 config.xml File

```
<config>
  <domain>
        <name>myDomain</name>
  </domain>
  <cluster>
      <server-name>myServer1</server-name>
      <multicast-address>239.255.0.1</multicast-address>
      <identity>1</identity>
      <enabled>evs4j</enabled>
  </cluster>
...
</config>
```

In Example 7–2, the server is part of a domain called `myDomain`.

For each server of the multi-server domain, the `multicast-address` elements must contain the same value. The `identity` and `server-name` elements, however, must be different for each server in the multi-server domain.

Example 7–2 shows the `config.xml` file of a second server, called `myServer2`, in the `myDomain` multi-server domain; note that this server's identity is 2.

Example 7–2 myServer2 config.xml File

```
<config>
```

```
  <domain>
        <name>myDomain</name>
  </domain>
  <cluster>
      <server-name>myServer2</server-name>
      <multicast-address>239.255.0.1</multicast-address>
      <identity>2</identity>
      <enabled>evs4j</enabled>
  </cluster>
...
</config>
```

See Section , "Order of cluster Element Child Elements" for a description of additional multi-server-related configuration elements and the required order of child elements.

4. Optionally, secure the messages that are shared between the servers in a domain by configuring encryption and digital signatures.

   See Section , "How to Secure the Messages Sent Between Servers in a Multi-Server Domain Using Oracle Event Processing Native Clustering".

5. To avoid a single point of failure, consider enabling Oracle Event Processing Visualizer on a small subset of `n` machines in the domain.

   See "How to Start Oracle Event Processing Visualizer in a Multi-Server Domain" in the *Oracle Fusion Middleware Visualizer User's Guide for Oracle Event Processing*.

   Note: Enabling Oracle Event Processing Visualizer on a given Oracle Event Processing Server may impact the performance of the server depending on the Oracle Event Processing Visualizer workload.

6. Start all servers in your multi-server domain.

   See Section , "Starting and Stopping an Oracle Event Processing Server in a Multi-Server Domain".

How to Create an Oracle Event Processing Multi-Server Domain With Custom Groups Using Oracle Event Processing Native Clustering

This procedure describes how to create a multi-server domain that uses custom groups to accommodate servers which are not completely homogenous.

If all the servers in your domain are completely homogeneous, you do not need to create custom groups. Instead, you can use the predefined groups: the singleton group and domain group. See Section , "How to Create an Oracle Event Processing Multi-Server Domain With Default Groups Using Oracle Event Processing Native Clustering".

For more information about default and custom groups, see Section , "Groups".

Note: In this section it is assumed that you have already created a domain that contains a single server and that you want to add additional servers to the domain to make it a multi-server domain. See Chapter 3, "Administering Oracle Event Processing Standalone-Server Domains" for details on creating a domain.

In this procedure, assume you have created three servers: `myServer1`, `myServer2`, and `myServer3`. You want `myServer1` to be a member of the `selector` group and `myServer2` and `myServer3` to be members of the `strategy` group.

To create an Oracle Event Processing multi-server domain with custom groups using Oracle Event Processing Native Clustering:

1. Create a domain that contains a single, initial server.

   See Chapter 3, "Administering Oracle Event Processing Standalone-Server Domains".

2. Add one or more servers to the domain using the Configuration Wizard.

   Note: Even though the Configuration WIzard does not support adding new servers to a multi-server domain, one can use the Configuration Wizard to generate a new stand-alone server, and then manually update its configuration to join a multi-server domain.

   See Section , "Updating an Oracle Event Processing Multi-Server Domain Using Oracle Event Processing Native Clustering."

3. Configure all the servers in the multi-server domain by manually editing their `config.xml` files and adding a `cluster` element with specific information.

   To configure the servers in a multi-server domain using custom groups, update the `config.xml` file for *each member server* by adding (if one does not already exist) a `groups` child element of `cluster` and specifying the name of the group as the value of the `groups` element.

   Note: When adding `cluster` element child elements, observe the correct element order as Section , "Order of cluster Element Child Elements" describes.

   The `groups` element can include more than one group name in the case that the server is a member of more than one group; separate multiple group names using commas.

   The `groups` element is optional; if a server configuration does not include one, then the server is a member of the default groups (domain and singleton). For more information about the domain and singleton groups, see Section , "Groups".

   Example 7–3, Example 7–4, and Example 7–5 show the relevant snippets of the `config.xml` file for each server.

Example 7–3 Server Configuration File config.xml for myServer1

```
<config>
  <domain>
        <name>myDomain</name>
  </domain>
  <cluster>
      <server-name>myServer1</server-name>
      <multicast-address>239.255.0.1</multicast-address>
      <identity>1</identity>
      <enabled>evs4j</enabled>
      <groups>selector</groups>
  </cluster>
```

```
...
</config>
```

Example 7–4 Server Configuration File config.xml for myServer2

```
<config>
  <domain>
        <name>myDomain</name>
  </domain>
  <cluster>
      <server-name>myServer2</server-name>
      <multicast-address>239.255.0.1</multicast-address>
      <identity>2</identity>
      <enabled>evs4j</enabled>
      <groups>strategy</groups>
  </cluster>
...
</config>
```

Example 7–5 Server Configuration File config.xml for myServer3

```
<config>
  <domain>
        <name>myDomain</name>
  </domain>
  <cluster>
      <server-name>myServer3</server-name>
      <multicast-address>239.255.0.1</multicast-address>
      <identity>3</identity>
      <enabled>evs4j</enabled>
      <groups>strategy</groups>
  </cluster>
...
</config>
```

4. Optionally, secure the messages that are shared between the servers in a domain by configuring encryption and digital signatures.

   See Section , "How to Secure the Messages Sent Between Servers in a Multi-Server Domain Using Oracle Event Processing Native Clustering".

5. To avoid a single point of failure, consider enabling Oracle Event Processing Visualizer on a small subset of n machines in the domain.

   See "How to Start Oracle Event Processing Visualizer in a Multi-Server Domain" in the *Oracle Fusion Middleware Visualizer User's Guide for Oracle Event Processing*.

   Note: Enabling Oracle Event Processing Visualizer on a given Oracle Event Processing Server may impact the performance of the server depending on the Oracle Event Processing Visualizer workload.

6. Start all servers in your multi-server domain.

   See Section , "Starting and Stopping an Oracle Event Processing Server in a Multi-Server Domain".

Updating an Oracle Event Processing Multi-Server Domain Using Oracle Event Processing Native Clustering

Use the Configuration Wizard to add a new server to an existing standalone server domain so as to later convert it into a multi-server domain. The procedure is similar to creating a new domain, so be sure you read Section , "Creating an Oracle Event Processing Standalone-Server Domain" before continuing with this section.

How to Update an Oracle Event Processing Multi-Server Domain Using the Configuration Wizard in Graphical Mode

The following procedure shows how to invoke and use the Configuration Wizard in graphical mode by executing the relevant command script for both Windows or Unix.

> **Note:** On Windows, you can also invoke the Configuration Wizard using the Start menu:
>
> `Start > All Programs > Oracle Event Processing 10gR3 > Tools > Configuration Wizard`

For clarity, it is assumed that:

- You have already created a new domain and its domain directory is `C:\oracle_cep\user_projects\domains\myDomain`.
- The domain includes a single server called `myServer1` and the server files are located in the `C:\oracle_cep\user_projects\domains\myDomain\myServer1` directory.
- You want to create a new server in the existing `myDomain` domain called `myServer2`.

To update an Oracle Event Processing multi-server domain using the Configuration Wizard in graphical mode:

1. Open a command window and set your environment as described in "Setting Your Development Environment" in the *Oracle Fusion Middleware Developer's Guide for Oracle Event Processing for Eclipse*.
2. Change to the *`ORACLE_CEP_HOME`*`/ocep_11.1/common/bin` directory, where *`ORACLE_CEP_HOME`* refers to the main Oracle Event Processing installation directory, such as `/oracle_cep`:

   ```
   prompt> cd /oracle_cep/ocep_11.1/common/bin
   ```

3. Invoke the `config.cmd` (Windows) or `config.sh` (UNIX) command to invoke the wizard:

   ```
   prompt> config.sh
   ```

   After the Configuration Wizard has finished loading, you will see a standard Oracle Welcome window.

   > **Note:** The Oracle Event Processing Configuration Wizard is self-explanatory; however, if you want more information about using the tool, continue reading this procedure.

4. Click **Next**.

5. In the Choose Create or Update Domain window, choose **Create a New Oracle Event Processing Domain**.
6. Click **Next**.
7. Enter the name and password of the administrator user for the new server you are adding to the domain.'
8. Click **Next**.
9. Enter basic configuration information about the new server in the domain.

   Note: If the new server is located on the same computer as any other servers in the domain, be sure the following information is different from that of the other servers to prevent conflicts when starting all servers

   In particular:

   - Enter the name of the new server.

     This name will also be used as the name of the directory that contains the new server's files. Following our example, this value is `myServer2`.

   - The listen port for the Oracle Event Processing server.
10. Click **Next**.
11. Enter and confirm the password for the Oracle Event Processing identity keystore.

    By default, the password for the certificate private key will be the same as the identity keystore; if you want it to be different, clear **Use Keystore Password** and enter the private key password.
12. Click **Next**.
13. In the Configuration Options window, choose:
    a. **Yes** if you want to change the default JDBC data source configuration,
    b. **No** to accept the defaults.

    Note: When you deploy an application to a group in the domain, Oracle Event Processing replicates the application to each server that is a member of the group. This means that if your application uses a datasource, and you have configured the datasource differently for each server in the domain, then the storage and retrieval of data to and from this data source will differ depending on the server on which the application is running.
14. Click **Next**.
15. If you chose to change the default JDBC data source configuration, enter the information in the Configure Database Properties window.

    In the top section, enter the name of the datasource. Then select the database type (Oracle or Microsoft SQL Server) and corresponding drivers; you can also browse to new drivers using the Browse/Append button.

    In the lower section, enter the details about the database to which this data source connects, such as its name, the name of the computer that hosts the database

server, the port, and the name and password of the user that connects to the database. The JDBC connection URL is automatically generated for you based on this information.

16. Click **Next**.
17. In the Configure Server window, enter the name of the *existing* domain and the full pathname of its location.

    Following our example, you would enter `myDomain` for the domain name and `C:\oracle_cep\user_projects\domains` for the domain location.
18. Click **Create**.
19. If the creation of the new server succeeded, you will see a message similar to the following in the Creating Domain window:

    ```
    Domain created successfully!
    Domain location: C:\oracle_cep\user_projects\domains\myDomain
    ```
20. Click **Done**.

Securing the Messages Sent Between Servers in a Multi-Server Domain

The servers in a multi-server domain update their state by exchanging multi-server-related messages. It is important that these messages be at least checked for integrity. A private key can be used to achieve integrity. This key must be shared by all servers in the domain.

This section describes:

- Section , "How to Secure the Messages Sent Between Servers in a Multi-Server Domain Using Oracle Event Processing Native Clustering"

How to Secure the Messages Sent Between Servers in a Multi-Server Domain Using Oracle Event Processing Native Clustering

You can secure the messages sent between servers in a multi-server domain using Oracle Event Processing native clustering.

To secure the messages sent between servers in an Oracle Event Processing multi-server domain using Oracle Event Processing Native Clustering:

1. Stop all servers in your multi-server domain, if they are currently running.

   See Section , "Starting and Stopping an Oracle Event Processing Server in a Multi-Server Domain".
2. Edit the `config.xml` file of each server in the multi-server domain by adding the `security` child element to the `cluster` element, as Example 7–6 shows.

Example 7–6 The cluster Element security Child Element

```
<config>
  <domain>
        <name>myDomain</name>
  </domain>
  <cluster>
      <server-name>myServer1</server-name>
      <multicast-address>239.255.0.1</multicast-address>
      <identity>1</identity>
      <enabled>coherence</enabled>
```

```
      <security>encrypt</security>
  </cluster>
...
</config>
```

The `config.xml` file is located in the *DOMAIN_DIR*`/`*servername*`/config` directory of each server, where *DOMAIN_DIR* refers to the domain directory and *servername* refers to the name of your server, such as `d:\oracle_cep_home\user_projects\domains\mydomain\myserver1\config`.

You must specify one of the following values for the `security` child element:

- `none`—Default value. Specifies that no security is configured for the multi-server domain.
- `encrypt`—Specifies that multi-server messages should be encrypted.

Observe the correct order of child elements in the cluster element. See Section , "Order of cluster Element Child Elements".

3. Edit the *DOMAIN_DIR*`/servername/config/security-config.xml` file of each server in the multi-server domain by adding the `encryption-service` child element of the `config` root element, as Example 7–7 shows.

Example 7–7 The security-config.xml File encryption-service Element

```
<config>
  <encryption-service>
     <signature-enabled>true</signature-enabled>
  </encryption-service>
  <css-realm>
   ...
</config>
```

4. Ensure that the *DOMAIN_DIR*`/`*servername*`/.aesinternal.dat` file for each server in the multi-server domain is exactly the same by copying the file from one server to the other servers.

   This file is automatically created by the Configuration Wizard when you first created the server; Oracle Event Processing uses this file for encrypting messages.

   For example, assume all the servers in your domain are located in the `d:\oracle_cep\user_projects\domains\mydomain` directory, and that the domain has three servers: `server1`, `server2`, and `server3`. To ensure they all have the same `.aesinternal.dat` file, copy the one from `server1` to the other servers:

   ```
   prompt> cd d:\oracle_cep\user_projects\domains\mydomain\server1
   prompt> cp .aesinternal.dat ..\server2
   prompt> cp .aesinternal.dat ..\server3
   ```

5. Start one of the servers in your domain.

   See Section , "Starting and Stopping an Oracle Event Processing Server in a Multi-Server Domain".

   Because of the `encryption-service` element that you added to the `security-config.xml` file in step 3, Oracle Event Processing automatically creates the `.msasig.dat` file in the main server directory. Oracle Event Processing uses this file for digitally signing messages.

6. Stop the server you just started.

See Section , "Starting and Stopping an Oracle Event Processing Server in a Multi-Server Domain".

7. Copy the `.msasig.dat` file you created in step 5 to the other servers.

   For example:

   ```
   prompt> cd d:\oracle_cep\user_projects\domains\mydomain\server1
   prompt> cp .msasig.dat ..\server2
   prompt> cp .msasig.dat ..\server3
   ```

8. If you plan to use Oracle Event Processing Visualizer with the servers in this domain, see Section , "How to Configure SSL in a Multi-Server Domain for Oracle Event Processing Visualizer".

9. Start all servers in your multi-server domain.

   See Section , "Starting and Stopping an Oracle Event Processing Server in a Multi-Server Domain".

Using the Multi-Server Domain APIs to Manage Group Membership Changes

In an active-active system, applications are deployed homogeneously across several servers and are actively executing.

There are cases, however, when these homogeneously-deployed applications need to elect a primary one as the coordinator or leader. In this case, events that result from the coordinator application are kept and passed on to the next component in the EPN; the results of secondary servers are dropped. However, if the coordinator fails, then one of the secondary servers must be elected as the new coordinator.

To enable this in an application, the adapter or event bean, generally in the role of an event sink, must implement the `com.bea.wlevs.ede.api.cluster.GroupMembershipListener` interface which allows the event sinks to listen for multi-server domain group membership changes. At runtime, Oracle Event Processing automatically invokes the `onMembershipChange` callback method whenever membership changes occur.

The signature of the callback method is as follows:

```
onMembershipChange(Server localIdentity, Configuration groupConfiguration);
```

In the implementation of the `onMembershipChange` callback method, the event sink uses the `Server` object (`localIdentity`) to verify if it is the leader. This can be done be comparing `localIdentity` with the result of `Configuration.getCoordinator()` run on the second parameter, `groupConfiguration`. This parameter also allows a server to know what the current members of the group are by executing `Configuration.getMembers()`.

In order to only keep events if it is a coordinator, the event sink must get a new `Server` identity every time membership in the group changes. Group membership changes occur if, for example, another server within the group fails and is no longer the coordinator.

A similar interface `com.bea.wlevs.ede.api.cluster.DomainMembershipListener` exists for listening to membership changes to the domain as a whole, rather than just changes to the group.

Note that in a hot-hot configuration, there is a non-zero delay in failure notification. If you are using the notification APIs to implement clustering, you will lose and not

process events that occur in the window between the server failure and the notification being delivered to the new master server.

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

Starting and Stopping an Oracle Event Processing Server in a Multi-Server Domain

To start the servers in a multi-server domain, start each server separately by running its start script. This is the same way you start a server in a standalone server domain. See Section , "Starting and Stopping an Oracle Event Processing Server in a Standalone-Server Domain" for details.

If you have not configured custom groups for the multi-server domain, then all servers are members of just the pre-defined domain group, which contains all the servers in the multi-server domain, and a singleton group, one for each member server. This means, for example, if there are three servers in the multi-server domain then there are three singleton groups.

If, however, you have configured custom groups for the multi-server domain, then the servers are members of the groups for which they have been configured, as well as the pre-defined groups.

8

Deploying Applications to Multi-Server Domains

This chapter describes how to deploy applications to Oracle Event Processing multi-server domains using Oracle Event Processing Visualizer or the deployer utility, as well as how to troubleshoot deployment.

This chapter includes the following sections:

Overview of Deploying an Application to an Oracle Event Processing Multi-Server Domain

When you deploy an application to a multi-server domain, you typically specify a target group, and Oracle Event Processing then deploys the application to the set of running servers in that group. Oracle Event Processing dynamically maintains group membership based on running servers. This means that if new servers in the group are started, Oracle Event Processing automatically propagates the appropriate set of deployments to the new server.

Take, for example, the simple multi-server domain configured in the section Section , "Creating an Oracle Event Processing Multi-Server Domain Using Oracle Coherence." Assume that only `myServer1` had been started, and then an application is deployed to the domain group, which includes `myServer1` and `myServer2`. At that point, because only `myServer1` of the multi-server domain has been started, the application will be deployed only to `myServer1`. When `myServer2` is subsequently started, Oracle Event Processing *automatically* replicates and propagates the deployment of the application to `myServer2` without the user having the explicitly deploy it.

Deployment propagation occurs based on application version. When you deploy a new version of an application, the new version is propagated to all servers accordingly.

If different configuration is required on different servers for an application then currently it is best to achieve this by using system properties.

This section describes how to perform the following tasks:

- Section , "Deploying to an Oracle Event Processing Server Using the Oracle Event Processing Visualizer"
- Section , "Deploying to an Oracle Event Processing Server Singleton Group Using the Deployer Utility"
- Section , "Deploying to an Oracle Event Processing Server Domain Group Using the Deployer Utility"
- Section , "Deploying to an Oracle Event Processing Server Custom Group Using the Deployer Utility"
- Section , "Troubleshooting Multi-Server Domain Deployment"

For more information, see:

- Section , "Groups"
- *Oracle Fusion Middleware Visualizer User's Guide for Oracle Event Processing*
- Appendix B, "Deployer Command-Line Reference"

Deploying to an Oracle Event Processing Server Using the Oracle Event Processing Visualizer

The simplest way to deploy an Oracle Event Processing application to a multi-server domain is to use the Oracle Event Processing Visualizer.

For more information, see "Deploying an Application" in the *Oracle Fusion Middleware Visualizer User's Guide for Oracle Event Processing*.

Deploying to an Oracle Event Processing Server Singleton Group Using the Deployer Utility

If you do not specify a group when you deploy an application, Oracle Event Processing deploys the application to the singleton server group that includes only the specific server to which you deploy the application. This is the standard case in single-server domains, but is also applicable to multi-server domains.

> **Note:** When you upgrade a 2.0 domain to execute in a multi-server domain, any deployed applications are deployed to the singleton server group.

The following example shows how to deploy to a singleton group; note that the command does not specify a `-group` option (in practice, the full command should be on one line):

```
prompt> java -jar wlevsdeploy.jar -url http://ariel:9002/wlevsdeployer
-install myapp_1.0.jar
```

In the example, the `myapp_1.0.jar` application will be deployed to the singleton server group that contains a single server: the one running on host `ariel` and listening to port `9002`. If the domain is multi-server and other servers are members of the domain group, the application will *not* be deployed to these servers.

For more information about groups, see Section , "Groups".

Deploying to an Oracle Event Processing Server Domain Group Using the Deployer Utility

The domain group is a live group that always exists and contains all servers in a domain. In another words, all servers are always a member of the domain group. However, you must still explicitly deploy applications to the domain group. The main reason for this is for simplicity and consistency in usage.

When you explicitly deploy an application to the domain group, Oracle Event Processing guarantees that all servers of this homogenous environment have this deployment.

To deploy to the domain group, use the `-group all` option. The following example shows how to deploy to a domain group:

```
prompt> java -jar wlevsdeploy.jar -url http://ariel:9002/wlevsdeployer -install
myapp_1.0.jar -group all
```

In the example, the `myapp_1.0.jar` application will be deployed to all servers of the domain group on host `ariel` listening to port `9002`.

For more information about groups, see Section , "Groups".

Deploying to an Oracle Event Processing Server Custom Group Using the Deployer Utility

To deploy to a custom group, use the `-group` *`groupname`* option of the deploy command.

In the following examples, assume the multi-server domain has been configured as described in Section , "How to Create an Oracle Event Processing Multi-Server Domain With Custom Groups Using Oracle Coherence."

The following example shows how to deploy an application called `strategies_1.0.jar` to the `strategygroup` (in practice, the full command should be on one line):

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

Based on the multi-server domain configuration, the preceding command deploys the application to `myServer2` and `myServer3`, the members of the group `strategygroup`.

The following example shows how to deploy an application called `selector_1.0.jar` to the `selectorgroup` (in practice, the full command should be on one line):

```
prompt> java -jar wlevsdeploy.jar -url http://ariel:9002/wlevsdeployer
-install selector_1.0.jar -group selectorgroup
```

Based on the multi-server domain configuration, the preceding command deploys the application only to `myServer1`, which is the sole member of group `selectorgroup`.

Note that both commands are executed to the same server (the one on host `ariel` listening to port `9002`). However, you can specify any of the servers in the domain in

the deploy command, even if the server is not part of the group to which you want to deploy the application.

For more information about groups, see Section , "Groups".

Troubleshooting Multi-Server Domain Deployment

This section describes common problems that you may encounter when deploying applications to an Oracle Event Processing multi-server domain, including:

- Section , "Oracle Event Processing Server Stops Application After Deployment"

Oracle Event Processing Server Stops Application After Deployment

Problem: After you deploy an application to an Oracle Event Processing multi-server domain, Oracle Event Processing stops the application after about 30 seconds.

Solution: Be sure you do not have more than one VPN software package installed on the same computer hosting your multi-server domain.

Part IV

Configuring Services

Part IV contains the following chapters:

9

Configuring Network I/O for Oracle Event Processing

This chapter describes how to configure network I/O in Oracle Event Processing, including how to configure either a network I/O server or client.

This chapter includes the following sections:

Overview of Network I/O in Oracle Event Processing

Oracle Event Processing supports network Input/Output (I/O) over Transmission Control Protocol/Internet Protocol (TCP/IP) using a variety of providers in both server and client mode.

You may define a network I/O service for both Secure Socket Layer (SSL) and non-SSL network access.

Oracle Event Processing supports both IPv4 and IPv6.

The following Oracle Event Processing services depend on network I/O configuration:

- `jetty`: depends on network I/O server (`netio`) configuration.
- `weblogic-rmi-client`: depends on network I/O client (`netio-client`) configuration.

For more information, see:

- Section , "Network I/O Providers"
- Section , "IPv4 and IPv6 Support"
- Section , "Configuring Network I/O Server (netio)"
- Section , "Configuring Network I/O Client (netio-client)"
- Chapter 10, "Configuring Security for Oracle Event Processing"
- Chapter 11, "Configuring Jetty for Oracle Event Processing"

Network I/O Providers

Table 9–1 lists the network I/O providers that Oracle Event Processing supports.

Table 9–1 Oracle Event Processing Network I/O Providers

provider-type	SSL?	Description
`non-blocking`	No	A non-blocking provider provides fully non-blocking I/O for reads and writes. That means that each call to `read` or `write` on the Connection interface will return immediately without blocking. If the underlying connection is not ready, then the `read` or `write` call will simply return zero. At that point, the calling code must use one of the notification mechanisms in the NetIO API to wait until the connection is ready to read or write. Non-Blocking providers may also support a non-blocking `connect` call, which means that a thread need not block if it takes a long time to establish (or fail to establish) a connection to a remote server.
`semi-blocking`	No	A semi-blocking provider provides non-blocking I/O for the `read` call, but each `write` call blocks until the data has been handed to the TCP/IP stack. Some platforms provide mechanisms that make it possible to implement a write-blocking provider that is faster than a fully non-blocking provider, but still allows for high scalability.
`blocking`	No	A blocking provider blocks on each `read` and `write` call until it is complete. If there is no data ready to read, then `read` will block until there is. This type of provider is much less scalable because there must be a thread waiting for each network connection that might have data. Oracle recommends that this type of provider should not typically be used.
`native`	No	The `NativeAsyncEngine` will be tried. If it is not supported, then an error will be raised.
`NIO`[1]	Yes	The `NIOEngine` will always be used.

[1] Default `provider-type`.

Example 9–1 shows how to specify a provider in the Oracle Event Processing server `config.xml` file `netio` element using the `provider-type` child element.

Example 9–1 Oracle Event Processing netio Element With provider-type Defined

```
<netio>
    <name>myNetio</name>
    <port>12345</port>
    <provider-type>non-blocking</provider-type>
</netio>
```

IPv4 and IPv6 Support

Oracle Event Processing server is certified for use with IPv4 only or IPv4/IPv6 dual-stack.

Oracle Event Processing does not support IPv6.

For more information about IPv6, see RFC 2460: Internet Protocol, Version 6 (IPv6) Specification (`http://www.ietf.org/Rift/rfc2460.txt`).

Configuring Network I/O Server (netio)

You can define a network I/O service that may be used by other services to act as the server and listen for incoming connections.

Alternatively, you can create a client network I/O service as Section , "Configuring Network I/O Client (netio-client)" describes.

How to Configure Network I/O Server

You configure network I/O server services using the `netio` element in the Oracle Event Processing server `config.xml` file.

For more information, see:

- Section , "Oracle Event Processing Server Configuration Files"
- Section , "Network I/O Providers"
- "netio" in the *Oracle Fusion Middleware Developer's Guide for Oracle Event Processing for Eclipse*

To configure network I/O server:

1. In the Oracle Event Processing server `config.xml` file, create a `netio` element as Example 9–2 shows.

Example 9–2 Oracle Event Processing netio Element

```
<netio>
</netio>
```

2. Add a `name` element that uniquely identifies this `netio` element on this Oracle Event Processing server as Example 9–3 shows.

Example 9–3 Oracle Event Processing netio Element With name Element

```
<netio>
    <name>MyNetIO</name>
</netio>
```

3. Add a `port` element to define the TCP/IP port on which this `netio` service will listen for connection requests as Example 9–4 shows.

Example 9–4 Oracle Event Processing netio Element With port Element

```
<netio>
    <name>MyNetIO</name>
    <port>9002</port>
</netio>
```

4. Optionally, specify a `provider-type` as Example 9–5 shows.

Example 9–5 Oracle Event Processing netio Element With port Element

```
<netio>
    <name>MyNetIO</name>
    <port>9002</port>
    <provider-type>NIO</provider-type>
</netio>
```

   For more information, see Section , "Network I/O Providers".

5. Optionally, specify the other `netio` child elements.

   For more information, see "netio" in the *Oracle Fusion Middleware Developer's Guide for Oracle Event Processing for Eclipse*.

Configuring Network I/O Client (netio-client)

You can define a network I/O service that may be used to perform non-blocking network I/O, but which will not act as a server and will not listen for incoming connections.

Alternatively, you can create a server network I/O service as Section , "Configuring Network I/O Server (netio)" describes.

How to Configure Network IO Client

You configure network I/O client services using the `netio-client` element in the Oracle Event Processing server `config.xml` file.

For more information, see:

- Section , "Oracle Event Processing Server Configuration Files"
- Section , "Network I/O Providers"
- "netio-client" in the *Oracle Fusion Middleware Developer's Guide for Oracle Event Processing for Eclipse*.

To configure network I/O client:

1. In the Oracle Event Processing server `config.xml` file, create a `netio-client` element as Example 9–2 shows.

Example 9–6 Oracle Event Processing netio-client Element

```
<netio-client>
</netio-client>
```

2. Add a `name` element that uniquely identifies this `netio` element on this Oracle Event Processing server as Example 9–3 shows.

Example 9–7 Oracle Event Processing netio-client Element With name Element

```
<netio-client>
    <name>MyNetIOClient</name>
</netio-client>
```

3. Optionally, specify a `provider-type` as Example 9–5 shows.

Example 9–8 Oracle Event Processing netio-client Element With port Element

```
<netio-client>
    <name>MyNetIOClient</name>
    <provider-type>NIO</provider-type>
</netio-client>
```

   For more information, see Section , "Network I/O Providers".

4. Optionally, specify the other `netio-client` child elements.

   For more information, see "netio-client" in the *Oracle Fusion Middleware Developer's Guide for Oracle Event Processing for Eclipse*.

10

Configuring Security for Oracle Event Processing

This chapter describes how to configure security in Oracle Event Processing, including configuring a security provider, SSL and FIPS, as well as configuring HTTPS-only connections and the security auditor.

This chapter includes the following sections:

Overview of Security in Oracle Event Processing

Oracle Event Processing 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.

To configure security for Oracle Event Processing server, consider the following general tasks:

1. Configure Java SE security.

   See Section , "Java SE Security".

2. Configure a security provider for authorization and authentication.

   See:

   - Section , "Security Providers"
   - Section , "Users, Groups, and Roles"

3. Configure password strength.

   See Section , "Configuring Password Strength".

4. Configure SSL and FIPS.

   See:

   - Section , "SSL"
   - Section , "FIPS"

5. Configure HTTPS-only connections.

   See Section , "Configuring HTTPS-Only Connections for Oracle Event Processing Server".

6. Configure security for individual Oracle Event Processing server services.

   See Section , "Configuring Security for Oracle Event Processing Server Services"

For more information, see:

- Section , "Enabling and Disabling Security"
- Section , "Security Utilities"
- Section , "Specifying User Credentials When Using the Command-Line Utilities"
- Section , "Security in Oracle Event Processing Examples and Domains"

Java SE Security

You can define Java SE security policies for:

- All the bundles that make up Oracle Event Processing
- Server startup
- Web applications deployed to the Oracle Event Processing server Jetty HTTP server
- Oracle Event Processing Visualizer

For more information, see:

- Section , "Configuring Java SE Security for Oracle Event Processing Server"
- `http://java.sun.com/javase/technologies/security/`

Security Providers

Oracle Event Processing supports various security providers for authentication, authorization, role mapping, and credential mapping.

Oracle Event Processing supports the following security providers:

- File-based—Default out-of-the-box security provider. This type of provider uses an operating system file to access security data such as user, password, and group information. Provides both authentication (process whereby identity of users is proved or verified) and authorization (process whereby a user's access to an Oracle Event Processing resource is permitted or denied based on the user's security role and the security policy assigned to the requested Oracle Event Processing resource). Authentication typically involves username/password combinations.

- LDAP—Provider that uses a Lightweight Data Access Protocol (LDAP) server to access user, password, and group information. Provides only authentication.
- DBMS—Provider that uses a database management system (DBMS) to access user, password, and group information. Provides both authentication and authorization.

If you choose to use the default file-based security provider, then you do not need to do any further configuration of your domain; the Configuration Wizard performs all necessary configuration. However, if you want to use the LDAP or DBMS providers, you must perform further configuration. See Section , "Configuring a Security Provider"

Once you have configured the security provider, you can start using Oracle Event Processing Visualizer to add new users, assign them to groups, and map groups to roles. See Section , "Users, Groups, and Roles".

Users, Groups, and Roles

Oracle Event Processing uses role-based authorization control to secure the Oracle Event Processing Visualizer and the `wlevs.Admin` command-line utility. There are a variety of default out-of-the-box security groups. You can add users to different groups to give them the different roles.

Administrators who use Oracle Event Processing Visualizer, `wlevs.Admin`, or any custom administration application that uses JMX to connect to an Oracle Event Processing instance use role-based authorization to gain access.

You can also use role-based authorization to control access to the HTTP publish-subscribe server.

There are two types of role:

- Application roles: application roles grant users the permission to access various Oracle CQL applications deployed to the Oracle Event Processing server. You can create application roles and associate them with the task roles that Oracle Event Processing provides.

  By default, administrator users can access any application and non-administration users cannot access any applications. Before a none-administration user can access an application, an administration user must grant the user the associated application role.
- Task roles: task roles grant users the permission to perform various tasks with the applications their application role authorizes them to access. Oracle Event Processing provides the default task roles that Table 10–1 describes.

Users that successfully authenticate themselves when using Oracle Event Processing Visualizer or `wlevs.Admin` are assigned roles based on their group membership, and then subsequent access to administrative functions is restricted according to the roles held by the user. Anonymous users (non-authenticated users) will not have any access to the Oracle Event Processing Visualizer or `wlevs.Admin`.

When an administrator uses the Configuration Wizard to create a new domain, they enter an administrator user that will be part of the `wlevsAdministrators` group. By default, this information is stored in a file-based provider filestore. The password is hashed using the SHA-256 algorithm. The default administrator user is named `wlevs` with password `wlevs`.

Table 10–1 describes the default Oracle Event Processing tasks roles available right after the creation of a new domain, as well as the name of the groups that are assigned to these roles.

Table 10–1 Default Oracle Event Processing Task Roles and Groups

Task Role	Group	Privileges
`Admin`	`wlevsAdministrators`	Has all privileges of all the preceding roles, as well as permission to: ■ Create users and groups ■ Configure HTTP publish-subscribe security ■ Change the system configuration, such as Jetty, work manager, and so on.
`ApplicationAdmin`	`wlevsApplicationAdmins`	Has all Operator privileges as well as permission to update the configuration of any deployed application.
`BusinessUser`	`wlevsBusinessUsers`	Has all Operator privileges as well as permission to update the Oracle CQL and EPL rules associated with the processor of a deployed application.
`Deployer`	`wlevsDeployers`	Has all Operator privileges as well as permission to deploy, undeploy, update, suspend, and resume any deployed application.
`Monitor`	`wlevsMonitors`	Has all Operator privileges as well as permission to enable/disable diagnostic functions, such as creating a diagnostic profile and recording events (then playing them back.)
`Operator`	`wlevsOperators`	Has read-only access to all server resources, services, and deployed applications.

Once the domain has been created, the administrator can use Oracle Event Processing Visualizer to create a group and associate it with one or more roles: each role grants access to an application. When you assign a user to a group, the roles you associate with the group give the user the privileges to access those applications.

For instructions on using Oracle Event Processing Visualizer to modify users, groups, and roles, see:

- "Managing Users" in the *Oracle Fusion Middleware Visualizer User's Guide for Oracle Event Processing*
- "Managing Groups" in the *Oracle Fusion Middleware Visualizer User's Guide for Oracle Event Processing*
- "Managing Roles" in the *Oracle Fusion Middleware Visualizer User's Guide for Oracle Event Processing*

For more information, see:

- Section , "Configuring HTTP Publish-Subscribe Server Channel Security"
- Section , "Specifying User Credentials When Using the Command-Line Utilities"
- Chapter 3, "Administering Oracle Event Processing Standalone-Server Domains"
- Chapter 5, "Administering Multi-Server Domains With Oracle Coherence"
- Chapter 7, "Administering Multi-Server Domains With Oracle Event Processing Native Clustering"

SSL

Oracle Event Processing provides one-way Secure Sockets Layer (SSL) to secure network traffic between Oracle Event Processing Visualizer and Oracle Event Processing server instances, between the Oracle Event Processing server instances of a multi-server domain, and between the `wlevs.Admin` command-line utility and Oracle Event Processing server instances.

You can configure Oracle Event Processing to use a Federal Information Processing Standards (FIPS)-certified pseudo-random number generator for SSL.

For more information, see:

- Section , "Configuring SSL to Secure Network Traffic"
- Section , "FIPS"
- Section , "Securing the Messages Sent Between Servers in a Multi-Server Domain"
- Oracle Coherence: Section , "Securing the Messages Sent Between Servers in a Multi-Server Domain"
- Oracle Event Processing Native Clustering: Section , "Running wlevs.Admin Utility in SSL Mode"

FIPS

The National Institute of Standards and Technology (NIST) creates standards for Federal computer systems. NIST issues these standards as Federal Information Processing Standards (FIPS) for use government-wide.

Oracle Event Processing supports FIPS using the `com.rsa.jsafe.provider.JsafeJCE` security provider. Using this provider, you can configure Oracle Event Processing to use a FIPS-certified pseudo-random number generator for SSL.

For more information, see:

- Section , "Configuring FIPS for Oracle Event Processing Server"
- Section , "SSL"
- `http://www.itl.nist.gov/fipspubs/`

Enabling and Disabling Security

After you configure SSL, you can configure the Oracle Event Processing server to accept only client requests on the HTTPS port. See Section , "Configuring HTTPS-Only Connections for Oracle Event Processing Server".

Optionally, you can disable security. See Section , "Disabling Security".

Security Utilities

Oracle Event Processing provides a variety of command-line utilities to simplify security administration. In addition to command-line utilities, you can use Oracle Event Processing Visualizer to perform many security tasks.

For more information, see:

- Appendix C, "Security Utilities Command-Line Reference"
- Section , "Specifying User Credentials When Using the Command-Line Utilities"

- "Security Tasks" in the *Oracle Fusion Middleware Visualizer User's Guide for Oracle Event Processing*

Specifying User Credentials When Using the Command-Line Utilities

Oracle Event Processing provides the following command-line utilities for performing a variety of tasks:

- `wlevs.Admin`: a command-line interface to administer Oracle Event Processing and, in particular, dynamically configure the rules for Oracle CQL and EPL processors and monitor the event latency and throughput of an application.

  See Appendix A, "wlevs.Admin Command-Line Reference" for details

- `Deployer`: a Java-based deployment utility that provides administrators and developers command-line based operations for deploying Oracle Event Processing applications.

  See Appendix B, "Deployer Command-Line Reference" for details.

- `cssconfig`: a command-line utility to generate a security configuration file (`security.xml`) that uses a password policy.

  See Appendix , "The cssconfig Command-Line Utility" for details.

- `encryptMSAConfig`: an encryption command-line utility to encrypt cleartext passwords, specified by the `password` element, in XML files.

  See Appendix , "The encryptMSAConfig Command-Line Utility" for details.

For each utility, you can specify user credentials (username and password) using the following three methods:

- On the command line using options such as `-user` and `-password`.
- Interactively so that the command line utility always prompts for the credentials.
- Specifying a filestore that stores the user credentials; the filestore itself is also password protected.

In a production environment you should *never* use the first option (specifying user credentials on the command line) but rather use only the second and third option.

When using interactive mode (command-line utility prompts for credentials), be sure you have the appropriate `terminalio` native libraries for your local computer in your `CLASSPATH` so that the user credentials are not echoed on the screen when you type them. Oracle Event Processing includes a set of standard native libraries for this purpose, but it may not include the specific one you need.

Security in Oracle Event Processing Examples and Domains

When you use the Configuration Wizard to create a new domain, you specify the administrator user and password, as well as the password to the domain identity keystore. This user is automatically added to the `wlevsAdministrators` group. All security configuration is stored using a file-based provider, by default.

All Oracle Event Processing examples are configured to have an administrator with username `wlevs` and password `wlevs`. When you create a new domain you specify the administrator name and password.

By default, security is disabled in the HelloWorld example. This means that any user can start the server, deploy applications, and run all commands of the administration tool (`wlevs.Admin`) without providing a password.

Security is enabled in the FX and AlgoTrading examples. In both examples, the user `wlevs`, with password `wlevs`, is configured to be the Oracle Event Processing administrator with full administrator privileges. The scripts to start the server for these examples use the appropriate arguments to pass this username and password to the `java` command. If you use the Deployer or `wlevs.Admin` utility, you must also pass this username/password pair using the appropriate arguments.

For more information, see Section , "Specifying User Credentials When Using the Command-Line Utilities".

Configuring Java SE Security for Oracle Event Processing Server

The Java SE platform defines a standards-based and interoperable security architecture that is dynamic and extensible. Security features — cryptography, authentication and authorization, public key infrastructure, and more — are built in.

Oracle Event Processing supports Java SE security by using the following security policies:

- `policy.xml`—Defines the security policies of all the bundles that make up Oracle Event Processing. The first bundle set defines the policies for server-related bundles; the second bundle set defines the policies for application bundles.
- `security.policy`—Defines the security policies for server startup and Web applications deployed to the Jetty HTTP server. This file also defines policies for the Oracle Event Processing Visualizer Web application.

Samples of the preceding files are shipped with the product and can be found in *ORACLE_CEP_HOME*`/ocep_11.1/utils/security`, where *ORACLE_CEP_HOME* refers to the directory in which you installed Oracle Event Processing, such as `/oracle_home`.

You can enable all Java SE security features with Oracle Event Processing.

For more information, see Section , "Java SE Security".

To configure Java SE security on the Oracle Event Processing server:

1. Stop the Oracle Event Processing server, if it is currently running.

   See Section , "Starting and Stopping Oracle Event Processing Servers".

2. Copy `policy.xml` and `security.policy`:

   - From: *ORACLE_CEP_HOME*`/ocep_11.1/utils/security`
   - To: *DOMAIN_DIR*`/`*servername*`/config`

   Where *ORACLE_CEP_HOME* refers to the directory in which you installed Oracle Event Processing (such as `/oracle_home`), *DOMAIN_DIR* refers to the main Oracle Event Processing installation directory, *servername* refers to the name of your server (such as `/oracle_cep/user_projects/domains/mydomain/myserver/config`).

3. Edit the two security policy files to suit your needs.

4. Update the server startup script for your platform located in the *DOMAIN_DIR*`/`*servername* directory, `startwlevs.cmd` (Windows) or `startwlevs.sh` (UNIX), by adding the following three properties to the `java` command that actually starts the server:

```
-Djava.security.manager
-Djava.security.policy=./config/security.policy
-Dcom.bea.core.security.policy=./config/policy.xml
```

For example (in practice, the full command should be on one line):

```
"%JAVA_HOME%\bin\java" %DGC% %DEBUG% -Djava.security.manager
-Djava.security.policy=./config/security.policy
-Dcom.bea.core.security.policy=./config/policy.xml
-Dwlevs.home="%USER_INSTALL_DIR%" -Dbea.hoe="%BEA_HOME%"
-jar "%USER_INSTALL_DIR%\bin\wlevs.jar" %1 %2 %3 %4 %5 %6
```

5. Update the *DOMAIN_DIR*/*servername*/config/config.xml file of your Oracle Event Processing server and edit the Jetty configuration by adding a <scratch-directory> child element of the <jetty> element to specify the directory to which Jetty Web applications are deployed. For example:

```
<jetty>
    <name>JettyServer</name>
    <network-io-name>NetIO</network-io-name>
    <work-manager-name>JettyWorkManager</work-manager-name>
    <secure-network-io-name>sslNetIo</secure-network-io-name>
    <scratch-directory>./JettyWork</scratch-directory>
</jetty>
```

6. Restart the Oracle Event Processing server for the changes to take effect.

   See Section , "Starting and Stopping Oracle Event Processing Servers".

Configuring a Security Provider

A security provider performs authentication, authorization, or both.

Oracle Event Processing server supports file-based, LDAP, and DBMS security providers.

The file-based security provider is the default security provider that the Configuration Wizard configures. If you want to use the file-based security provider, no further configuration is required.

The LDAP security provider supports authentication only.

The DBMS security provider supports both authentication and authorization.

This section describes:

- Section , "Configuring Authentication Using the LDAP Provider and Authorization Using the DBMS Provider"
- Section , "Configuring Both Authentication and Authorization Using the DBMS Provider"

For more information, see Section , "Security Providers".

Configuring Authentication Using the LDAP Provider and Authorization Using the DBMS Provider

The following procedure describes how to configure the LDAP security provider for authentication and the DBMS provider for authorization.

> **Caution:** When using LDAP for authentication, you can not add or delete users and groups using Oracle Event Processing Visualizer, you can only change the password of a user.

To configure authentication using the LDAP provider and Authorization using the DBMS provider:

1. Open a command window and set your environment as described in "Setting Your Development Environment" in the *Oracle Fusion Middleware Developer's Guide for Oracle Event Processing for Eclipse*.

2. Add the *ORACLE_CEP_HOME*\ocep_11.1\bin directory to your PATH environment variable, where *ORACLE_CEP_HOME* is the main Oracle Event Processing installation directory, such as d:\oracle_cep:

```
prompt> set PATH=d:\oracle_cep\ocep_11.1\bin;%PATH% (Windows)
prompt> PATH=/oracle_cep/ocep_11.1/bin:$PATH (UNIX)
```

3. Change to the *DOMAIN_DIR*/*servername*/config directory, where *DOMAIN_DIR* refers to the main directory of your domain, such as d:\oracle_cep\user_projects\domains\mydomain, and *servername* refers to the name of your server:

```
prompt> cd d:\oracle_cep\user_projects\domains\mydomain\defaultserver\config
```

4. Using your favorite text editor, create a file called myLDAPandDBMS.properties and copy into it the entire contents of Example 10–1.

Example 10–1 LDAP/DBMS Properties File

```
# For attributes of type boolean or Boolean, value can be "true" or "false"
# and it's case insensitive.
# For attributes of type String[], values are comma separated; blanks before
# and after the comma are ignored. For example, if the property is defined as:
#   saml1.IntersiteTransferURIs=uri1, uri2, uri3
# the IntersiteTransferURIs attribute value is String[]{"uri1", "uri2", "uri3"}
# For attributes of type Properties, the value should be inputted as
# a set of key=value pairs separated by commas; blanks before and after the
# commas are also ignored. For example (in practice, the property should be all on one line):
#  store.StoreProperties=DriverName=oracle.jdbc.driver.OracleDriver,
ConnectionURL=jdbc:oracle:thin:@united.bea.com:1521:xe, Username=user, Password=user
domain.mbean=com.bea.common.management.configuration.LegacyDomainInfoMBean
domain.DomainName=legacy-domain-name
domain.ServerName=legacy-server-name
domain.RootDirectory=legacy-rootdir
#domain.ProductionModeEnabled=
#domain.WebAppFilesCaseInsensitive=
domain.DomainCredential=changeit
jaxp.mbean=com.bea.common.management.configuration.JAXPFactoryServiceMBean
#jaxp.DocBuilderFactory=
#jaxp.SaxParserFactory=
#jaxp.SaxTransformFactory=
#jaxp.TransformFactory=
#ldapssl.mbean=com.bea.common.management.configuration.LDAPSSLSocketFactoryLookupServiceMBean
#ldapssl.Protocol=
#ldapssl.TrustManagerClassName=
namedsql.mbean=com.bea.common.management.configuration.NamedSQLConnectionLookupServiceMBean
store.mbean=com.bea.common.management.configuration.StoreServiceMBean
# Split here for readability; in practice, a property should be all on one line.
store.StoreProperties=DriverName=oracle.jdbc.driver.OracleDriver,
    ConnectionURL=jdbc:oracle:thin:@localhost:1521:orcl, Username=wlevs, Password=wlevs
#store.ConnectionProperties=
#store.NotificationProperties=
realm.mbean=weblogic.management.security.RealmMBean
realm.Name=my-realm
#realm.ValidateDDSecurityData=
#realm.CombinedRoleMappingEnabled=
#realm.EnableWebLogicPrincipalValidatorCache=
#realm.MaxWebLogicPrincipalsInCache=
#realm.DelegateMBeanAuthorization=
```

```
#realm.AuthMethods=
adt.1.mbean=weblogic.security.providers.audit.DefaultAuditorMBean
adt.1.Severity=INFORMATION
#adt.1.InformationAuditSeverityEnabled=
#adt.1.WarningAuditSeverityEnabled=
#adt.1.ErrorAuditSeverityEnabled=
#adt.1.SuccessAuditSeverityEnabled=
#adt.1.FailureAuditSeverityEnabled=
#adt.1.OutputMedium=
#adt.1.RotationMinutes=
#adt.1.BeginMarker=
#adt.1.EndMarker=
#adt.1.FieldPrefix=
#adt.1.FieldSuffix=
adt.1.Name=my-auditor
#adt.1.ActiveContextHandlerEntries=
atn.1.mbean=weblogic.security.providers.authentication.LDAPAuthenticatorMBean
#atn.1.UserObjectClass=
#atn.1.UserNameAttribute=
#atn.1.UserDynamicGroupDNAttribute=
atn.1.UserBaseDN=o=ECS,dc=bea,dc=com
atn.1.UserSearchScope=subtree
#atn.1.UserFromNameFilter=
#atn.1.AllUsersFilter=
atn.1.GroupBaseDN=ECS,dc=bea,dc=com
#atn.1.GroupSearchScope=
#atn.1.GroupFromNameFilter=
#atn.1.AllGroupsFilter=
#atn.1.StaticGroupObjectClass=
#atn.1.StaticGroupNameAttribute=
atn.1.StaticMemberDNAttribute=member
#atn.1.StaticGroupDNsfromMemberDNFilter=
#atn.1.DynamicGroupObjectClass=
#atn.1.DynamicGroupNameAttribute=
#atn.1.DynamicMemberURLAttribute=
atn.1.GroupMembershipSearching=unlimited
atn.1.MaxGroupMembershipSearchLevel=0
atn.1.UseRetrievedUserNameAsPrincipal=false
#atn.1.IgnoreDuplicateMembership=
#atn.1.KeepAliveEnabled=
atn.1.Credential=wlevs
#atn.1.Name=
#atn.1.PropagateCauseForLoginException=
atn.1.ControlFlag=REQUIRED
#atn.1.ConnectTimeout=
atn.1.Host=localhost
atn.1.Port=389
#atn.1.SSLEnabled=
atn.1.Principal=cn=Administrator,dc=bea,dc=com
#atn.1.CacheEnabled=
#atn.1.CacheSize=
#atn.1.CacheTTL=
atn.1.FollowReferrals=false
#atn.1.BindAnonymouslyOnReferrals=
#atn.1.ResultsTimeLimit=
#atn.1.ParallelConnectDelay=
#atn.1.ConnectionRetryLimit=
atn.1.EnableGroupMembershipLookupHierarchyCaching=true
#atn.1.MaxGroupHierarchiesInCache=
#atn.1.GroupHierarchyCacheTTL=
#atn.5.mbean=weblogic.security.providers.authentication.OpenLDAPAuthenticatorMBean
#atn.5.UserNameAttribute=
#atn.5.UserBaseDN=
#atn.5.UserFromNameFilter=
#atn.5.GroupBaseDN=
#atn.5.GroupFromNameFilter=
```

```
#atn.5.StaticGroupObjectClass=
#atn.5.StaticMemberDNAttribute=
#atn.5.StaticGroupDNsfromMemberDNFilter=
#atn.5.UserObjectClass=
#atn.5.UserDynamicGroupDNAttribute=
#atn.5.UserSearchScope=
#atn.5.AllUsersFilter=
#atn.5.GroupSearchScope=
#atn.5.AllGroupsFilter=
#atn.5.StaticGroupNameAttribute=
#atn.5.DynamicGroupObjectClass=
#atn.5.DynamicGroupNameAttribute=
#atn.5.DynamicMemberURLAttribute=
#atn.5.GroupMembershipSearching=
#atn.5.MaxGroupMembershipSearchLevel=
#atn.5.UseRetrievedUserNameAsPrincipal=
#atn.5.IgnoreDuplicateMembership=
#atn.5.KeepAliveEnabled=
#atn.5.Credential=
#atn.5.PropagateCauseForLoginException=
#atn.5.ControlFlag=
#atn.5.Name=
#atn.5.ConnectTimeout=
#atn.5.Host=
#atn.5.Port=
#atn.5.SSLEnabled=
#atn.5.Principal=
#atn.5.CacheEnabled=
#atn.5.CacheSize=
#atn.5.CacheTTL=
#atn.5.FollowReferrals=
#atn.5.BindAnonymouslyOnReferrals=
#atn.5.ResultsTimeLimit=
#atn.5.ParallelConnectDelay=
#atn.5.ConnectionRetryLimit=
#atn.5.EnableGroupMembershipLookupHierarchyCaching=
#atn.5.MaxGroupHierarchiesInCache=
#atn.5.GroupHierarchyCacheTTL=
cm.1.mbean=weblogic.security.providers.credentials.DefaultCredentialMapperMBean
cm.1.Name=my-credential-mapper
cm.1.CredentialMappingDeploymentEnabled=true
#cm.3.mbean=weblogic.security.providers.credentials.FileBasedCredentialMapperMBean
#cm.3.FileStorePath=
#cm.3.FileStorePassword=
#cm.3.EncryptAlgorithm=
#cm.3.Name=
#cm.3.CredentialMappingDeploymentEnabled=
rm.1.mbean=weblogic.security.providers.xacml.authorization.XACMLRoleMapperMBean
rm.1.Name=my-role-mapper
rm.1.RoleDeploymentEnabled=true
atz.1.mbean=weblogic.security.providers.xacml.authorization.XACMLAuthorizerMBean
atz.1.Name=my-authorizer
atz.1.PolicyDeploymentEnabled=true
adj.1.mbean=weblogic.security.providers.authorization.DefaultAdjudicatorMBean
adj.1.RequireUnanimousPermit=false
adj.1.Name=my-adjudicator
```

Customize the property file by updating the `store.StoreProperties` property to reflect your database driver information, connection URL, and username and password of the user that connects to the database. This is how the default property is set:

```
# Split for readability; in practice, the property should be on one line.
store.StoreProperties=DriverName=oracle.jdbc.driver.OracleDriver,
ConnectionURL=jdbc:oracle:thin:@mymachine:1521:orcl, Username=wlevs,
Password=wlevs
```

Also update the property that specifies your LDAP server configuration.

Leave all the other properties to their default values.

5. Make a backup copy of the existing `security.xml` file, in case you need to revert:

```
prompt> copy security.xml security.xml_save
```

6. Create a new security configuration file (`security.xml`) by executing the following `cssconfig` command:

```
prompt> cssconfig -p myLDAPandDBMS.properties -c security.xml -i
security-key.dat
```

In the preceding command, `myLDAPandDBMS.properties` is the property file you created in step 4, `security.xml` is the name of the new security configuration file, and `security-key.dat` is an existing file, generated by the Configuration Wizard, that contains the identity key.

See Section , "The cssconfig Command-Line Utility" for additional information.

7. Change to the *ORACLE_CEP_HOME*`/ocep_11.1/utils/security/sql` directory:

```
prompt> cd d:\oracle_cep\ocep_11.1\utils\security\sql
```

This directory contains SQL scripts for creating the required security-related database tables and populating them with initial data. Because you are using the DBMS provider only for authorization, the relevant scripts for this procedure are:

- `atz_create.sql`—Creates all tables required for authorization.
- `atz_drop.sql`—Drops all authorization-related tables.

8. Run the following SQL script against the database you specified as the database store in step 4:

- `atz_create.sql`

9. Configure your LDAP server by adding the default groups described in Section , "Users, Groups, and Roles" as well as the administrator user you specified when you created the domain. By default, this user is called `wlevs`.

Refer to your LDAP server documentation for details.

10. Optionally, configure password strength in your new `security.xml` file.

See Section , "Configuring Password Strength".

Configuring Both Authentication and Authorization Using the DBMS Provider

The following procedure describes how to configure the DBMS security provider for both authentication and authorization.

To configure both authentication and authorization using the DBMS provider:

1. Open a command window and set your environment as described in "Setting Your Development Environment" in the *Oracle Fusion Middleware Developer's Guide for Oracle Event Processing for Eclipse*.

2. Add the *ORACLE_CEP_HOME*`\ocep_11.1\bin` directory to your `PATH` environment variable, where *ORACLE_CEP_HOME* is the main Oracle Event Processing installation directory, such as `d:\oracle_cep`:

```
prompt> set PATH=d:\oracle_cep\ocep_11.1\bin;%PATH% (Windows)
```

```
prompt> PATH=/oracle_cep/ocep_11.1/bin:$PATH (UNIX)
```

3. Change to the *DOMAIN_DIR*/*servername*/config directory, where *DOMAIN_DIR* refers to the main directory of your domain, such as d:\oracle_cep\user_projects\domains\mydomain, and *servername* refers to the name of your server:

```
prompt> cd d:\oracle_cep\user_projects\domains\mydomain\defaultserver\config
```

4. Make a backup copy of the existing security.xml file, in case you need to revert:

```
prompt> copy security.xml security.xml_save
```

5. Using your favorite text editor, create a file called myDBMS.properties and copy into it the entire contents of Example 10–2.

Example 10–2 DBMS Property File

```
# For attributes of type boolean or Boolean, value can be "true" or "false"
# and it's case insensitive.
# For attributes of type String[], values are comma separated; blanks before
# and after the comma are ignored. For example, if the property is defined as:
#   saml1.IntersiteTransferURIs=uri1, uri2, uri3
# the IntersiteTransferURIs attribute value is String[]{"uri1", "uri2", "uri3"}
# For attributes of type Properties, the value should be inputted as
# a set of key=value pairs separated by commas; blanks before and after the
# commas are also ignored. For example (split for readability; in practice, the property
should be all on one line):
#  store.StoreProperties=DriverName=oracle.jdbc.driver.OracleDriver,
    ConnectionURL=jdbc:oracle:thin:@united.bea.com:1521:xe, Username=user, Password=user
domain.mbean=com.bea.common.management.configuration.LegacyDomainInfoMBean
domain.DomainName=legacy-domain-name
domain.ServerName=legacy-server-name
domain.RootDirectory=legacy-rootdir
#domain.ProductionModeEnabled=
#domain.WebAppFilesCaseInsensitive=
domain.DomainCredential=changeit
jaxp.mbean=com.bea.common.management.configuration.JAXPFactoryServiceMBean
#jaxp.DocBuilderFactory=
#jaxp.SaxParserFactory=
#jaxp.SaxTransformFactory=
#jaxp.TransformFactory=
#ldapssl.mbean=com.bea.common.management.configuration.LDAPSSLSocketFactoryLookupServiceMBean
#ldapssl.Protocol=
#ldapssl.TrustManagerClassName=
namedsql.mbean=com.bea.common.management.configuration.NamedSQLConnectionLookupServiceMBean
store.mbean=com.bea.common.management.configuration.StoreServiceMBean
# Split for readability; the property should be fully on one line.
store.StoreProperties=DriverName=oracle.jdbc.driver.OracleDriver,
    ConnectionURL=jdbc:oracle:thin:@mymachine:1521:orcl, Username=wlevs, Password=wlevs
#store.ConnectionProperties=
#store.NotificationProperties=
realm.mbean=weblogic.management.security.RealmMBean
realm.Name=my-realm
#realm.ValidateDDSecurityData=
#realm.CombinedRoleMappingEnabled=
#realm.EnableWebLogicPrincipalValidatorCache=
#realm.MaxWebLogicPrincipalsInCache=
#realm.DelegateMBeanAuthorization=
#realm.AuthMethods=
sqlconn.1.mbean=com.bea.common.management.configuration.NamedSQLConnectionMBean
sqlconn.1.Name=POOL1
sqlconn.1.JDBCDriverClassName=oracle.jdbc.driver.OracleDriver
sqlconn.1.ConnectionPoolCapacity=5
sqlconn.1.ConnectionPoolTimeout=10000
sqlconn.1.AutomaticFailoverEnabled=false
```

```
sqlconn.1.PrimaryRetryInterval=0
sqlconn.1.JDBCConnectionURL=jdbc\:oracle\:thin\:@fwang02\:1521\:orcl
sqlconn.1.JDBCConnectionProperties=
sqlconn.1.DatabaseUserLogin=wlevs
sqlconn.1.DatabaseUserPassword=wlevs
sqlconn.1.BackupJDBCConnectionURL=
sqlconn.1.BackupJDBCConnectionProperties=
sqlconn.1.BackupDatabaseUserLogin=
sqlconn.1.BackupDatabaseUserPassword=
adt.1.mbean=weblogic.security.providers.audit.DefaultAuditorMBean
adt.1.Severity=INFORMATION
#adt.1.InformationAuditSeverityEnabled=
#adt.1.WarningAuditSeverityEnabled=
#adt.1.ErrorAuditSeverityEnabled=
#adt.1.SuccessAuditSeverityEnabled=
#adt.1.FailureAuditSeverityEnabled=
#adt.1.OutputMedium=
#adt.1.RotationMinutes=
#adt.1.BeginMarker=
#adt.1.EndMarker=
#adt.1.FieldPrefix=
#adt.1.FieldSuffix=
adt.1.Name=my-auditor
#adt.1.ActiveContextHandlerEntries=
atn.1.mbean=weblogic.security.providers.authentication.SQLAuthenticatorMBean
atn.1.PasswordAlgorithm=SHA-1
atn.1.PasswordStyle=SALTEDHASHED
atn.1.PasswordStyleRetained=true
atn.1.SQLCreateUser=INSERT INTO USERS VALUES ( ? , ? , ? )
atn.1.SQLRemoveUser=DELETE FROM USERS WHERE U_NAME \= ?
atn.1.SQLRemoveGroupMemberships=DELETE FROM GROUPMEMBERS WHERE G_MEMBER \= ? ORG_NAME \= ?
atn.1.SQLSetUserDescription=UPDATE USERS SET U_DESCRIPTION  \= ? WHERE U_NAME \= ?
atn.1.SQLSetUserPassword=UPDATE USERS SET U_PASSWORD \= ? WHERE U_NAME \= ?
atn.1.SQLCreateGroup=INSERT INTO GROUPS VALUES ( ? , ? )
atn.1.SQLSetGroupDescription=UPDATE GROUPS SET G_DESCRIPTION \= ? WHERE G_NAME \=  ?
atn.1.SQLAddMemberToGroup=INSERT INTO GROUPMEMBERS VALUES( ?, ?)
atn.1.SQLRemoveMemberFromGroup=DELETE FROM GROUPMEMBERS WHERE G_NAME \= ? AND G_MEMBER \= ?
atn.1.SQLRemoveGroup=DELETE FROM GROUPS WHERE G_NAME \= ?
atn.1.SQLRemoveGroupMember=DELETE FROM GROUPMEMBERS WHERE G_NAME \= ?
atn.1.SQLListGroupMembers=SELECT G_MEMBER FROM GROUPMEMBERS WHERE G_NAME \= ? AND G_MEMBER
LIKE ?
atn.1.DescriptionsSupported=true
atn.1.SQLGetUsersPassword=SELECT U_PASSWORD FROM USERS WHERE U_NAME \= ?
atn.1.SQLUserExists=SELECT U_NAME FROM USERS WHERE U_NAME \= ?
atn.1.SQLListMemberGroups=SELECT G_NAME FROM GROUPMEMBERS WHERE G_MEMBER \= ?
atn.1.SQLListUsers=SELECT U_NAME FROM USERS WHERE U_NAME LIKE ?
atn.1.SQLGetUserDescription=SELECT U_DESCRIPTION FROM USERS WHERE U_NAME \= ?
atn.1.SQLListGroups=SELECT G_NAME FROM GROUPS WHERE G_NAME LIKE ?
atn.1.SQLGroupExists=SELECT G_NAME FROM GROUPS WHERE G_NAME \= ?
atn.1.SQLIsMember=SELECT G_MEMBER FROM GROUPMEMBERS WHERE G_NAME \= ? AND G_MEMBER \= ?
atn.1.SQLGetGroupDescription=SELECT G_DESCRIPTION FROM GROUPS WHERE G_NAME \= ?
atn.1.GroupMembershipSearching=unlimited
atn.1.MaxGroupMembershipSearchLevel=0
atn.1.DataSourceName=POOL1
atn.1.PlaintextPasswordsEnabled=true
atn.1.ControlFlag=REQUIRED
atn.1.Name=my-authenticator
atn.1.EnableGroupMembershipLookupHierarchyCaching=false
atn.1.MaxGroupHierarchiesInCache=100
atn.1.GroupHierarchyCacheTTL=60
cm.1.mbean=weblogic.security.providers.credentials.DefaultCredentialMapperMBean
cm.1.Name=my-credential-mapper
cm.1.CredentialMappingDeploymentEnabled=true
rm.1.mbean=weblogic.security.providers.xacml.authorization.XACMLRoleMapperMBean
rm.1.Name=my-role-mapper
rm.1.RoleDeploymentEnabled=true
```

```
atz.1.mbean=weblogic.security.providers.xacml.authorization.XACMLAuthorizerMBean
atz.1.Name=my-authorizer
atz.1.PolicyDeploymentEnabled=true
adj.1.mbean=weblogic.security.providers.authorization.DefaultAdjudicatorMBean
adj.1.RequireUnanimousPermit=false
adj.1.Name=my-adjudicator
```

Customize the property file by updating the `store.StoreProperties` property to reflect your database driver information, connection URL, and username and password of the user that connects to the database. This is how the default property is set (in practice, this setting should be on one line):

```
store.StoreProperties=DriverName=oracle.jdbc.driver.OracleDriver,
ConnectionURL=jdbc:oracle:thin:@mymachine:1521:orcl, Username=wlevs,
Password=wlevs
```

Leave all the other properties to their default values.

6. Create a new security configuration file (`security.xml`) by executing the following `cssconfig` command:

```
prompt> cssconfig -p myDBMS.properties -c security.xml -i security-key.dat
```

In the preceding command, `myDBMS.properties` is the property file you created in step 4, `security.xml` is the name of the new security configuration file, and `security-key.dat` is an existing file, generated by the Configuration Wizard, that contains the identity key.

See Section , "The cssconfig Command-Line Utility" for additional information.

7. Change to the *`ORACLE_CEP_HOME`*`/ocep_11.1/utils/security/sql` directory:

```
prompt> cd d:\oracle_cep\ocep_11.1\utils\security\sql
```

This directory contains SQL scripts for creating the required security-related database tables and populating them with initial data. These scripts are:

- `atn_create.sql`—Creates all tables required for authentication.
- `atn_drop.sql`—Drops all authentication-related tables.
- `atn_init.sql`—Inserts default values into the authentication-related user and group tables. In particular, the script inserts a single default administrator user called `wlevs`, with password `wlevs`, into the user table and specifies that the user belongs to the wlevsAdministrators group. The script also inserts the default groups listed in Table 10–1 into the group table.
- `atz_create.sql`—Creates all tables required for authorization.
- `atz_drop.sql`—Drops all authorization-related tables.

8. If, when you created your domain using the Configuration Wizard, you specified an administrator user *other* than the default `wlevs`, edit the `atn_init.sql` file and add the `INSERT INTO USERS` and corresponding `INSERT INTO GROUPMEMBERS` statements accordingly.

For example, to add an administrative user `juliet`, with password `shackell`, add the following statements to the `atn_init.sql` file:

```
INSERT INTO USERS (U_NAME, U_PASSWORD, U_DESCRIPTION) VALUES
('juliet','shackell','default admin');
INSERT INTO GROUPMEMBERS (G_NAME, G_MEMBER) VALUES
('wlevsAdministrators','juliet');
```

9. Run the following SQL script files, in the order listed, against the database you specified as the database store in step 4:
   - `atn_create.sql`
   - `atn_init.sql`
   - `atz_create.sql`
10. Optionally, configure password strength in your new `security.xml` file.

    See Section , "Configuring Password Strength".

Configuring Password Strength

Password strength is a measurement of the effectiveness of a password as an authentication credential. How the password strength is configured determines the type of password a user can specify, such as whether the password can contain the username, the minimum length of the password, the minimum number of numeric characters it can contain, and so on.

You configure the strength of the passwords used for Oracle Event Processing authentication by updating the security configuration file (`security.xml`), located in the *`DOMAIN_DIR`*`/`*`servername`*`/config` directory, where *`DOMAIN_DIR`* refers to your domain directory, such as `d:/oracle_cep/user_projects/domains/mydomain`, and *`servername`* refers to your server, such as `defaultserver`.

The password strength configuration is contained in the `<password-validator>` element.

Example 10–3 shows a snippet from the `security.xml` file with the default values after creating a new domain using the Configuration Wizard.

Example 10–3 Default password-validator Element in the security.xml File

```
<sec:password-validator
   xmlns:pas="http://www.bea.com/ns/weblogic/90/security/providers/passwordvalidator"
   xsi:type="pas:system-password-validatorType">
   <sec:name>my-password-validator</sec:name>
   <pas:reject-equal-or-contain-username>true</pas:reject-equal-or-contain-username>
   <pas:reject-equal-or-contain-reverse-username>
      false
   </pas:reject-equal-or-contain-reverse-username>
   <pas:max-password-length>50</pas:max-password-length>
   <pas:min-password-length>6</pas:min-password-length>
   <pas:max-instances-of-any-character>0</pas:max-instances-of-any-character>
   <pas:max-consecutive-characters>0</pas:max-consecutive-characters>
   <pas:min-alphabetic-characters>1</pas:min-alphabetic-characters>
   <pas:min-numeric-characters>1</pas:min-numeric-characters>
   <pas:min-lowercase-characters>1</pas:min-lowercase-characters>
   <pas:min-uppercase-characters>1</pas:min-uppercase-characters>
   <pas:min-non-alphanumeric-characters>0</pas:min-non-alphanumeric-characters>
</sec:password-validator>
```

Table 10–2 describes all the child elements of `<password-validator>` you can configure.

If you manually update the `security.xml` file, you must restart the Oracle Event Processing server instance for the changes to take effect.

Table 10–2 Child Elements of <password-validator>

Child Element	Description	Default Value
`reject-equal-or-contain-name`	When set to `true`, Oracle Event Processing rejects a password if it is the same as, or contains, the username. When set to `false`, Oracle Event Processing does not reject a password for this reason.	`true`
`reject-equal-or-contain-reverse-username`	When set to `true`, Oracle Event Processing rejects a password if it is the same as, or contains, the reversed username. When set to `false`, Oracle Event Processing does not reject a password for this reason.	`false`
`max-password-length`	Specifies the maximum length of a password. A value of 0 means there is no restriction. Valid values for this element are integers greater than or equal to 0.	50
`min-password-length`	Specifies the minimum length of a password. Valid values for this element are integers greater than or equal to 0.	6
`max-instances-of-any-character`	Specifies the maximum number of times the same character can appear in the password. For example, if this element is set to 2, then the password `bubble` is invalid. A value of 0 means there is no restriction. Valid values for this element are integers greater than or equal to 0.	0
`max-consecutive-characters`	Specifies the maximum number of repeating consecutive characters that are allowed in the password. For example, if this element is set to 2, then the password `bubbble` is invalid. A value of 0 means there is no restriction. Valid values for this element are integers greater than or equal to 0.	0
`min-alphabetic-characters`	Specifies the minimum number of alphabetic characters that a password must contain. A value of 0 means there is no restriction. Valid values for this element are integers greater than or equal to 0.	1
`min-numeric-characters`	Specifies the minimum number of numeric characters that a password must contain. A value of 0 means there is no restriction. Valid values for this element are integers greater than or equal to 0.	1
`min-lowercase-characters`	Specifies the minimum number of lowercase characters that a password must contain. A value of 0 means there is no restriction. Valid values for this element are integers greater than or equal to 0.	0
`min-uppercase-characters`	Specifies the minimum number of uppercase characters that a password must contain. A value of 0 means there is no restriction. Valid values for this element are integers greater than or equal to 0.	0
`min-non-alphanumeric-characters`	Specifies the minimum number of non-alphanumeric characters that a password must contain. Non-alphanumeric characters include `$`, `#`, `@`, `&`, `!` and so on. A value of 0 means there is no restriction. Valid values for this element are integers greater than or equal to 0.	0

Configuring SSL to Secure Network Traffic

Oracle Event Processing uses one-way Secure Sockets Layer (SSL) to secure the network traffic between:

- A browser running the Oracle Event Processing Visualizer and the Oracle Event Processing instance that hosts the data-services application that the Oracle Event Processing Visualizer uses.
- The `wlevs.Admin` command-line utility and an Oracle Event Processing instance.

  See Section , "Running wlevs.Admin Utility in SSL Mode".
- The member servers of a multi-server domain.

You configure SSL in the server's `config.xml` file. When you create an Oracle Event Processing server using the Configuration Wizard, the server's `config.xml` automatically includes a default SSL configuration.

This section describes:

- Section , "How to Configure SSL Manually"
- Section , "How to Create a Key-Store Manually"
- Section , "How to Configure SSL in a Multi-Server Domain for Oracle Event Processing Visualizer"

For more information, see Section , "SSL".

How to Configure SSL Manually

This section describes how to configure SSL in Oracle Event Processing.

To configure SSL manually:

1. Create a domain using the Configuration Wizard.

   See:

   - Section , "Creating an Oracle Event Processing Standalone-Server Domain"
   - Section , "Creating an Oracle Event Processing Multi-Server Domain Using Oracle Coherence"
   - Section , "Creating an Oracle Event Processing Multi-Server Domain Using Oracle Event Processing Native Clustering"
2. Using your favorite XML editor, open the Oracle Event Processing server `config.xml` file.

   By default, the Configuration Wizard creates the `config.xml` file in the *`ORACLE_CEP_HOME`*`/user_projects/domains/`*`DOMAIN_DIR`*`/`*`servername`*`/config` directory, where *`ORACLE_CEP_HOME`* refers to the Oracle Event Processing installation directory (such as `d:/oracle_cep`), *`DOMAIN_DIR`* refers to the domain directory (such as `my_domain`), and *`servername`* refers to the server instance directory (such as `server1`).

   For more information, see Section , "Oracle Event Processing Server Configuration Files".
3. Configure the `ssl` element.

   Example 10–4 shows the default `ssl` element the Configuration Wizard creates.

Example 10–4 Default ssl Element

```
<ssl>
    <name>sslConfig</name>
    <key-store>./ssl/evsidentity.jks</key-store>
    <key-store-pass>
        <password>{Salted-3DES}sdlUX8aEDeNpQ4VhsaCnFA==</password>
    </key-store-pass>
    <key-store-alias>evsidentity</key-store-alias>
    <key-manager-algorithm>SunX509</key-manager-algorithm>
    <ssl-protocol>TLS</ssl-protocol>
    <enforce-fips>false</enforce-fips>
    <need-client-auth>false</need-client-auth>
</ssl>
```

The `key-store` element points to a certificate file. The Configuration Wizard creates a default certificate file, called `evsidentity.jks`, in the *`DOMAIN_DIR`*`/`*`servername`*`/ssl` directory; its password is the same as that entered when creating a server with the Configuration Wizard.

By default, the password for the certificate private key will be the same as the password for the identity keystore.

> **Note:** The Oracle Event Processing Server will not start unless the password for certificate private key is the same as the password for the identity keystore.

The `evsidentity.jks` contains a self-signed certificate. Optionally, create your own certificate file and either replace the `evsidentity.jks` file, or update the `key-store` element in the `config.xml` file.

> **Note:** In a production environment, the system administrator should replace the default self-signed certificate with a CA signed certificate.

For more information on creating a key-store yourself, see Section , "How to Create a Key-Store Manually".

For more information on the `enforce-fips` element, see Section , "Configuring FIPS for Oracle Event Processing Server".

4. Configure a `netio` element for SSL.

   Example 10–5 shows the default `netio` element the Configuration Wizard creates.

Example 10–5 Default netio Element

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

The `ssl-config-bean-name` must match the `ssl` element `name` child element (see step 3).

Optionally, change this port to a port number that suits your needs.

The default secure port is `9003` by default.

5. Configure the `jetty` element to add a `secure-network-io-name` child element.

   Example 10–6 shows the default jetty element the Configuration Wizard creates.

Example 10–6 Default jetty Element

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

   The `secure-network-io-name` must match the SSL `netio` element `name` child element (see step 4).

6. Save and close the `config.xml` file.
7. Restart the Oracle Event Processing server (if running).

   See Section , "Starting and Stopping Oracle Event Processing Servers".

How to Create a Key-Store Manually

By default, the Configuration Wizard creates a default key-store certificate file, called `evsidentity.jks`, in the *`DOMAIN_DIR`*`/`*`servername`*`/ssl` directory; its password is the same as that entered when creating a server with the Configuration Wizard. Optionally, you can manually create your own key-store.

For more information, see:

- Section , "Creating an Oracle Event Processing Standalone-Server Domain Using the Configuration Wizard in Graphical Mode"
- Section , "How to Configure SSL Manually"

To create a key-store manually:

1. Use the JDK keytool command to generate a key-store:

   ```
   keytool -genkey -alias evsidentity -keyalg RSA -validity 10958 -keystore
   evsidentity.jks -keysize 1024
   ```

2. Enter the key-store password, as prompted:

   ```
   Enter keystore password:
   ```

3. Enter the key-store attributes, as prompted:

   ```
   What is your first and last name?
     [Unknown]:  CEP
   What is the name of your organizational unit?
     [Unknown]:  SOA
   What is the name of your organization?
     [Unknown]:  ORACLE
   What is the name of your City or Locality?
     [Unknown]:  SF
   What is the name of your State or Province?
     [Unknown]:  CA
   What is the two-letter country code for this unit?
     [Unknown]:  US
   Is CN=CEP, OU=SOA, O=ORACLE, L=SF, ST=CA, C=US correct?
     [no]:  y
   ```

4. When prompted for a key password, do not enter a password; just press RETURN:

```
Enter key password for <evsidentity>
        (RETURN if same as keystore password):
```

> **Note:** The Oracle Event Processing Server will not start unless the password for certificate private key is the same as the password for the identity keystore.

5. Using your favorite XML editor, open the Oracle Event Processing server `config.xml` file.

   By default, the Configuration Wizard creates the `config.xml` file in the *ORACLE_CEP_HOME*`/user_projects/domains/`*DOMAIN_DIR*`/`*servername*`/config` directory, where *ORACLE_CEP_HOME* refers to the Oracle Event Processing installation directory (such as `d:/oracle_cep`), *DOMAIN_DIR* refers to the domain directory (such as `my_domain`), and *servername* refers to the server instance directory (such as `server1`).

   For more information, see Section , "Oracle Event Processing Server Configuration Files".

6. Configure the `ssl` element.

   Example 10–4 shows the default `ssl` element the Configuration Wizard creates.

Example 10–7 Default ssl Element

```
<ssl>
    <name>sslConfig</name>
    <key-store>KEYSTORE_PATH</key-store>
    <key-store-pass>
        <password>PASSWORD</password>
    </key-store-pass>
    <key-store-alias>KEYSTORE_ALIAS</key-store-alias>
    <key-manager-algorithm>SunX509</key-manager-algorithm>
    <ssl-protocol>TLS</ssl-protocol>
    <enforce-fips>false</enforce-fips>
    <need-client-auth>false</need-client-auth>
</ssl>
```

   Where:

   - *KEYSTORE_PATH* is the file path to the key-store file (the file name is from the `-keystore` argument to the keytool command).
   - *PASSWORD* is the cleartext keystore password.
   - *KEYSTORE_ALIAS* is the keystore alias (from the `-alias` argument to the keytool command).

7. Save and close the `config.xml` file.

8. Encrypt the cleartext password in the `key-store-pass` element `password` child element of the `config.xml` file by using the `encryptMSAConfig` utility.

   See Section , "The encryptMSAConfig Command-Line Utility."

How to Configure SSL in a Multi-Server Domain for Oracle Event Processing Visualizer

The following procedure shows how to configure one-way SSL between the server that hosts the Oracle Event Processing Visualizer data-services application and another server in a multi-server domain.

In the procedure, it is assumed that the server that hosts the Oracle Event Processing Visualizer data-services application is called `server1` and the other server is called `server2`, and that both are located in the `/oracle_cep/user_projects/domains/mydomain` directory. Repeat this procedure for other servers in the domain, if required.

For information on securing the messages sent between servers in a multi=-server domain, see:

- Oracle Coherence: Section , "Securing the Messages Sent Between Servers in a Multi-Server Domain"
- Oracle Event Processing Native Clustering: Section , "Securing the Messages Sent Between Servers in a Multi-Server Domain"

For information on starting Oracle Event Processing Visualizer in a multi-server domain, see "How to Start Oracle Event Processing Visualizer in a Multi-Server Domain" in the *Oracle Fusion Middleware Visualizer User's Guide for Oracle Event Processing*.

To configure SSL in a multi-server domain for use by Oracle Event Processing Visualizer:

1. Ensure that SSL is configured for the two servers in the domain.

   If you used the Configuration Wizard to create the servers, then SSL is configured by default.

   See Section , "How to Configure SSL Manually" for details, as well as information on how to change the default configuration.

2. Start `server2`.

   See Section , "Starting and Stopping Oracle Event Processing Servers".

3. Open a command window and set your environment as described in "Setting Your Development Environment" in the *Oracle Fusion Middleware Developer's Guide for Oracle Event Processing for Eclipse*.

4. Change to the `ssl` sub-directory of the `server1` directory:

   ```
   prompt> cd /oracle_cep/user_projects/domains/mydomain/server1/ssl
   ```

5. Generate a trust keystore for `server1` that includes the certificate of `server2` by specifying the following command (split for readability; in practice, the command should be on one line):

   ```
   prompt> java -classpath ORACLE_CEP_HOME\ocep_
   11.1\common\lib\evspath.jar;ORACLE_CEP_HOME\ocep_
   11.1\utils\security\wlevsgrabcert.jar
   com.bea.wlevs.security.util.GrabCert host:secureport
   -alias=alias truststorepath
   ```

   where

   - *ORACLE_CEP_HOME* refers to the Oracle Event Processing installation directory (such as `d:/oracle_cep`)

- *host* refers to the computer on which server2 is running.
- *secureport* refers to the SSL network i/o port configured for server2. Default value is 9003.

  For more information, see Example 10–5 in Section , "How to Configure SSL Manually."
- *alias* refers to the alias for the certificate in the trust keystore. Default value is the hostname.
- *truststorepath* refers to the full pathname of the generated trust keystore file; default is evstrust.jks

For example (split for readability; in practice, the command should be on one line):

```
prompt> java -classpath C:\OracleCEP\ocep_
11.1\common\lib\evspath.jar;C:\OracleCEP\ocep_
11.1\utils\security\wlevsgrabcert.jar
com.bea.wlevs.security.util.GrabCert server2:9003
-alias=server2 evstrust.jks
```

For more information, see Section , "The GrabCert Command-Line Utility".

6. When prompted, enter the Oracle Event Processing administrator password:

```
Please enter the Password for the supplied user : wlevs
```

7. When prompted, select the certificate sent by server2:

```
Created TrustStore evstrust.jks
Opening connection to server2:9003...
Starting SSL handshake...

No certificates in evstrust.jks are trusted by server2:9003

Server sent 1 certificate(s):

 1 Subject CN=localhost, OU=Event Server, O=BEA, L=San Jose, ST=California,
C=US
   Issuer  CN=localhost, OU=Event Server, O=BEA, L=San Jose, ST=California,
C=US
   sha1    00 07 c0 f4 10 48 9a f9 07 82 4f b6 9c 7f 7c d0 37 57 90 7d
   md5     a4 d4 ff d2 43 69 95 ca c3 43 e6 f6 b8 08 df b7

Enter certificate to add to trusted keystore evstrust.jks or 'q' to quit: [1]
```

8. Update the config.xml file of server1, adding trust keystore information to the ssl element and adding a use-secure-connections element, as shown in bold in the following snippet:

```
<ssl>
    <name>sslConfig</name>
    <key-store>./ssl/evsidentity.jks</key-store>
    <key-store-pass>
        <password>{Salted-3DES}s4YUEvH4Wl2DAjb45iJnrw==</password>
    </key-store-pass>
    <key-store-alias>evsidentity</key-store-alias>
    <key-manager-algorithm>SunX509</key-manager-algorithm>
    <ssl-protocol>TLS</ssl-protocol>
    <trust-store>./ssl/evstrust.jks</trust-store>
    <trust-store-pass>
        <password>wlevs</password>
```

```
        </trust-store-pass>
        <trust-store-alias>evstrust</trust-store-alias>
        <trust-store-type>JKS</trust-store-type>
        <trust-manager-algorithm>SunX509</trust-manager-algorithm>
        <enforce-fips>false</enforce-fips>
        <need-client-auth>false</need-client-auth>
    </ssl>
    <use-secure-connections>
        <value>true</value>
    </use-secure-connections>
```

The config file is located in the `config` subdirectory of the main server directory, such as `/oracle_cep/user_projects/domains/mydomain/server1/config/`.

9. Encrypt the cleartext password in the `trust-store-pass` element `password` child element of the `config.xml` file by using the `encryptMSAConfig` utility.

   See Section , "The encryptMSAConfig Command-Line Utility."

10. Start `server1`.

Configuring FIPS for Oracle Event Processing Server

You can configure Oracle Event Processing server to use a Federal Information Processing Standards (FIPS)-certified pseudo-random number generator.

For more information, see Section , "FIPS".

To configure FIPS for Oracle Event Processing server:

1. Configure Java SE security.

   See Section , "Configuring Java SE Security for Oracle Event Processing Server".

2. Configure SSL.

   See Section , "Configuring SSL to Secure Network Traffic".

3. Copy com.bea.core.jsafejcefips_*version*.jar:

   - From: *ORACLE_CEP_HOME*`/ocep_11.1/utils/security`
   - To: *JRE_HOME*`/jre/lib/ext`

   Where *ORACLE_CEP_HOME* refers to the directory in which you installed Oracle Event Processing and *JRE_HOME* refers to the directory that contains your JRockit JRE:

   a. If using the JRockit JDK installed with Oracle JRockit Real Time, copy the com.bea.core.jsafejcefips_*version*.jar into the *JROCKIT_HOME*`/JROCKIT_RT_HOME/jre/lib/ext` directory.

      Where *JROCKIT_HOME* is the directory in which you installed Oracle JRockit Real Time, such as `d:\jrockit`.

   b. If using the JRockit JDK installed with Oracle Event Processing, copy the com.bea.core.jsafejcefips_*version*.jar into the *ORACLE_CEP_HOME/JROCKIT_HOME*`/jre/lib/ext` directory.

      Where *ORACLE_CEP_HOME* is the directory in which you installed Oracle Event Processing server such as `d:\oracle_cep`.

4. Stop the Oracle Event Processing server, if it is currently running.

   See Section , "Starting and Stopping Oracle Event Processing Servers".

5. Edit the *JRE_HOME*/jre/lib/security/java.security file to add com.bea.core.jsafejcefips_2.0.0.0.jar as a JCE provider as Example 10–8 shows.

Example 10–8 Editing java.security to Add jsafejcefips JAR as a JCE Provider

```
security.provider.N=com.rsa.jsafe.provider.JsafeJCE
```

Where *N* is a unique integer that specifies the order in which Java accesses security providers.

To make the JsafeJCE provider the default provider, set *N* to 1. In this case, change the value of *N* for any other providers in the java.security file so that each provider has a unique number as Example 10–9 shows.

Example 10–9 Making JsafeJCE the Default Provider

```
security.provider.1=com.rsa.jsafe.provider.JsafeJCE
security.provider.2=sun.security.provider.Sun
```

6. Edit the server.config file ssl element as Example 10–10 shows to add the following child elements:
   - enforce-fips: set this option to true.
   - secure-random-algorithm: set this option to FIPS186PRNG
   - secure-random-provider: set this option to JsafeJCE.

Example 10–10 Editing server.config to Enable Fips

```
<ssl>
    <name>sslConfig</name>
    <key-store>./ssl/evsidentity.jks</key-store>
    <key-store-pass>
        <password>s4YUEvH4Wl2DAjb45iJnrw==</password>
    </key-store-pass>
    <key-store-alias>evsidentity</key-store-alias>
    <key-manager-algorithm>SunX509</key-manager-algorithm>
    <ssl-protocol>TLS</ssl-protocol>
    <enforce-fips>true</enforce-fips>
    <need-client-auth>false</need-client-auth>
    <secure-random-algorithm>FIPS186PRNG</secure-random-algorithm>
    <secure-random-provider>JsafeJCE</secure-random-provider>
</ssl>
```

7. Restart the Oracle Event Processing server for the changes to take effect.

   See Section , "Starting and Stopping Oracle Event Processing Servers".

Configuring HTTPS-Only Connections for Oracle Event Processing Server

This section describes how to lock down the server so that only HTTPS connections are allowed.

To configure HTTPS-Only connections for Oracle Event Processing server:

1. Ensure that SSL is configured for the server.

   See Section , "Configuring SSL to Secure Network Traffic" for details.

2. Remove the HTTP port configuration from the server's *DOMAIN_DIR*/*servername*/config/config.xml file, leaving only the configuration for the HTTPS port.

   Example 10–11 shows a config.xml snippet with a standard configuration in which both an HTTP and HTTPS port have been configured. The HTTP port is 9002 and the HTTPS port is 9003. Clients can access the Jetty server using both ports.

Example 10–11 Typical config.xml File With Both HTTP and HTTPS Access

```
<netio>
   <name>NetIO</name>
   <port>9002</port>
</netio>
<netio>
   <name>sslNetIo</name>
   <port>9003</port>
   <ssl-config-bean-name>sslConfig</ssl-config-bean-name>
</netio>
<jetty>
   <name>JettyServer</name>
   <network-io-name>NetIO</network-io-name>
   <secure-network-io-name>sslNetIo</secure-network-io-name>
   ...
</jetty>
<ssl>
   <name>sslConfig</name>
   <key-store>./ssl/evsidentity.jks</key-store>
   ...
</ssl>
```

   Example 10–12 shows the same config.xml file with HTTP access removed. Clients can now access the Jetty server only using the HTTPS port.

Example 10–12 Typical config.xml File With HTTP Access Removed

```
<netio>
   <name>sslNetIo</name>
   <port>9003</port>
   <ssl-config-bean-name>sslConfig</ssl-config-bean-name>
</netio>
<jetty>
   <name>JettyServer</name>
   <secure-network-io-name>sslNetIo</secure-network-io-name>
   ...
</jetty>
<ssl>
   <name>sslConfig</name>
   <key-store>./ssl/evsidentity.jks</key-store>
   ...
</ssl>
```

3. If you have a multi-server domain, be sure that SSL has been configured between the member servers.

   See Section , "How to Configure SSL in a Multi-Server Domain for Oracle Event Processing Visualizer" for details.

Configuring Security for Oracle Event Processing Server Services

After you complete basic security tasks such as configuring Java SE security, a security service provider, and SSL, you can configure security details specific to the various services that Oracle Event Processing server provides.

This section describes:

- Section , "Configuring Jetty Security"
- Section , "Configuring JMX Security"
- Section , "Configuring JDBC Security"
- Section , "Configuring HTTP Publish-Subscribe Server Channel Security"

Configuring Jetty Security

Oracle Event Processing supports Jetty (see `http://www.eclipse.org/jetty/`) as Java Web server to deploy HTTP servlets and static resources.

The following security tasks affect Jetty configuration:

- Section , "Configuring Java SE Security for Oracle Event Processing Server"
- Section , "Configuring SSL to Secure Network Traffic"

For more information on Jetty, see Chapter 11, "Configuring Jetty for Oracle Event Processing".

Configuring JMX Security

Clients that access the Oracle Event Processing server using JMX are subject to Oracle Event Processing role-based authentication.

For more information, see:

- Section , "Users, Groups, and Roles"
- "Managing Groups" in the *Oracle Fusion Middleware Visualizer User's Guide for Oracle Event Processing*
- "Managing Users" in the *Oracle Fusion Middleware Visualizer User's Guide for Oracle Event Processing*

For more information about JMX, see Chapter 12, "Configuring JMX for Oracle Event Processing".

Configuring JDBC Security

If you update a `data-source` with a new password using the Configuration Wizard, the Configuration Wizard performs password encryption for you.

If you update the `config.xml` file manually by adding or modifying a `data-source` element, you enter the password in plain text and then encrypt the password using the encryption utility `encryptMSAConfig`.

Example 10–13 shows a `config.xml` file `data-source` element with a new plain text password `secret` specified in the `properties` element with name `password`.

Example 10–13 Oracle Event Processing config.xml File data-source Element After Encryption

```
<data-source>
```

```
   <name>epcisDS</name>
   <driver-params>
      <url>jdbc:sqlserver://localhost:1433;databaseName=myDB;SelectMethod=cursor</url>
      <driver-name>com.microsoft.sqlserver.jdbc.SQLServerDriver</driver-name>
      <properties>
         <element>
            <name>user</name>
            <value>juliet</value>
         </element>
         <element>
            <name>password</name>
            <value>secret</value>
         </element>
      </properties>
   </driver-params>
</data-source>
<transaction-manager>
   <name>TM</name>
   <rmi-service-name>RMI</rmi-service-name>
</transaction-manager>
```

Example 10–14 shows the config.xml file data-source element after encryption. Note the plain text password has been encrypted.

Example 10–14 Oracle Event Processing config.xml File data-source Element After Encryption

```
<data-source>
   <name>epcisDS</name>
   <driver-params>
      <url>jdbc:sqlserver://localhost:1433;databaseName=myDB;SelectMethod=cursor</url>
      <driver-name>com.microsoft.sqlserver.jdbc.SQLServerDriver</driver-name>
      <properties>
         <element>
            <name>user</name>
            <value>juliet</value>
         </element>
         <element>
            <name>password</name>
            <value>{Salted-3DES}hVgC5iZ3nZA=</value>
         </element>
      </properties>
   </driver-params>
</data-source>
<transaction-manager>
   <name>TM</name>
   <rmi-service-name>RMI</rmi-service-name>
</transaction-manager>
```

For more information, see:

- Section , "Specifying User Credentials When Using the Command-Line Utilities"
- Section , "The encryptMSAConfig Command-Line Utility"

For more information about JDBC, see Chapter 13, "Configuring JDBC for Oracle Event Processing"

Configuring HTTP Publish-Subscribe Server Channel Security

After you configure at least one HTTP publish-subscribe server channel, you can use role-based authentication to control access to individual HTTP publish-subscribe server channels using the Oracle Event Processing Visualizer.

For more information, see:

- Section , "Users, Groups, and Roles"
- Chapter 14, "Configuring HTTP Publish-Subscribe for Oracle Event Processing"
- "Configuring Security for the HTTP Publish-Subscribe Channels" in the *Oracle Fusion Middleware Visualizer User's Guide for Oracle Event Processing*.

Configuring Cross-Domain Security for Oracle Event Processing Visualizer

Oracle Event Processing Visualizer provides an Adobe Flash-based user interface with which you can create and configure event processing networks. In order to provide the most flexible default performance for Oracle Event Processing Visualizer, the software is installed with a configured trust level that allows access to Visualizer data from any domain. If you find that this trust level is inappropriate for your deployment, you can edit the application's Flash cross-domain policy file in order to restrict access.

You should review the domains which are allowed by the Flash cross-domain policy and determine whether it is appropriate for the application to fully trust both the intentions and security posture of those domains.

You'll find a more thorough description on editing cross-domain policy at the Adobe web site. For more information on using Adobe cross-domain policy files, see the Adobe security web site.

Updating cross-domain security involves opening the Oracle Event Processing Visualizer JAR file. Here are the high-level steps:

1. Locate the Oracle Event Processing Visualizer JAR file. By default in an Oracle Event Processing installation, you'll find it at:

   CEP_HOME/modules/com.bea.wlevs.visualizer.jmxhttpadapter_*version*.jar

   For example, on a Windows installation, that might be:

   C:\Oracle\Middleware\ocep_
   11.1\modules\com.bea.wlevs.visualizer.jmxhttpadapter_11.1.1.6_0.jar
2. Expand the JAR file to locate crossdomain.war.
3. Expand crossdomain.war to locate crossdomain.xml.
4. Edit crossdomain.xml to reflect your cross-domain security needs.
5. Repackage crossdomain.war and the Oracle Event Processing Visualizer JAR file.

Configuring the Oracle Event Processing Security Auditor

Oracle Event Processing provides a security auditor that logs security-related activity.

By default, the security auditor logs to `DOMAIN_DIR/servername/legacy-rootdir/servers/legacy-server-name/logs/DefaultAuditRecorder.log` file, where `DOMAIN_DIR` refers to the main directory of your domain, such as `d:\oracle_cep\user_projects\domains\mydomain`, and `servername` refers to the name of your server.

By default, the Oracle Event Processing security auditor will only log security errors or failures. This helps keep the security auditor log file at a manageable size.

Optionally, you can configure the level at which the Oracle Event Processing security auditor logs information.

For more information, see "Configuring the WebLogic Auditing Provider" in the *Oracle Fusion Middleware Securing Oracle WebLogic Server*.

To configure security auditor logging:

1. Change to the *DOMAIN_DIR*/*servername*/config directory, where *DOMAIN_DIR* refers to the main directory of your domain, such as d:\oracle_cep\user_projects\domains\mydomain, and *servername* refers to the name of your server:

   ```
   prompt> cd d:\oracle_cep\user_projects\domains\mydomain\defaultserver\config
   ```

2. Using your favorite text editor, edit the security.xml file.
3. Locate the sec:auditor element.

   Example 10–15 shows the default sec:auditor element configuration:

Example 10–15 Default sec:auditor Element

```
<sec:auditor xsi:type="wls:default-auditorType">
    <sec:name>my-auditor</sec:name>
    <wls:severity>CUSTOM</wls:severity>
    <wls:rotation-minutes>720</wls:rotation-minutes>
    <wls:error-audit-severity-enabled>true</wls:error-audit-severity-enabled>
    <wls:failure-audit-severity-enabled>true</wls:failure-audit-severity-enabled>
</sec:auditor>
```

4. Modify the sec:auditor element as required:
   - wls:rotation-minutes: Specifies how many minutes to wait before creating a new DefaultAuditRecorder.log file. At the specified time, the audit file is closed and a new one is created. A backup file named DefaultAuditRecorder.YYYYMMDDHHMM.log (for example, DefaultAuditRecorder.200405130110.log) is created in the same directory.
   - wls:severity: Specifies the severity level appropriate for your Oracle Event Processing server as Table 10–3 lists. The Oracle Event Processing security auditor audits security events of the specified severity, as well as all events with a higher numeric severity rank. For example, if you set the severity level to ERROR, the Oracle Event Processing security auditor audits security events of severity level ERROR, SUCCESS, and FAILURE.

Table 10–3 Oracle Event Processing Security Auditor Severity Levels

Event Severity	Rank
INFORMATION	1
WARNING	2
ERROR	3
SUCCESS	4
FAILURE	5

   You can also set the wls:severity level to CUSTOM, and then enable (set to true) or disable (set to false) the specific severity levels you want to audit using one or more of the following child elements as Example 10–15 shows:
   - wls:information-audit-severity-enabled: If the severity value is set to CUSTOM, setting this child element to true causes the Oracle Event

Processing security auditor to generate audit records for events with a severity level of `INFORMATION`.

- `wls:warning-audit-severity-enabled`: If the `severity` value is set to `CUSTOM`, setting this child element to true causes the Oracle Event Processing security auditor to generate audit records for events with a severity level of `WARNING`.
- `wls:error-audit-severity-enabled`: If the `severity` value is set to `CUSTOM`, setting this child element to true causes the Oracle Event Processing security auditor to generate audit records for events with a severity level of `ERROR`.
- `wls:success-audit-severity-enabled`: If the `severity` value is set to `CUSTOM`, setting this child element to true causes the Oracle Event Processing security auditor to generate audit records for events with a severity level of `SUCCESS`.
- `wls:failure-audit-severity-enabled`: If the `severity` value is set to `CUSTOM`, setting this child element to true causes the Oracle Event Processing security auditor to generate audit records for events with a severity level of `FAILURE`.

5. Save and close the `security.xml` file.
6. Restart the Oracle Event Processing server for the changes to take effect.

   See Section , "Starting and Stopping Oracle Event Processing Servers".

Disabling Security

You can disable security entirely on the Oracle Event Processing server. While this configuration may be appropriate for development environments, Oracle does not recommend disabling security in a production environment.

To temporarily disable security, you can run the `startwlevs.cmd` or `startwlevs.sh` script with the `-disablesecurity` argument on the command line. For example:

```
startwlevs.cmd -disablesecurity
```

Note: In some sample domains, the `startwlevs.cmd` and `startwlevs.sh` scripts already include a `-disablesecurity` argument. Executing such a script with `-disablesecurity` on the command line will fail with an `Illegal argument` error.

11

Configuring Jetty for Oracle Event Processing

This chapter describes how to configure Jetty for use with Oracle Event Processing, including configuring network I/O, and work managers, as well as configuring a Jetty server instance.

This chapter includes the following sections:

Overview of Jetty Support in Oracle Event Processing

Oracle Event Processing supports Jetty (see `http://www.eclipse.org/jetty/`) as Java Web server to deploy HTTP servlets and static resources.

Oracle Event Processing support for Jetty is based on Version 1.2 the OSGi HTTP Service. This API provides ability to dynamically register and unregister `http://java.sun.com/products/servlet/docs.html` objects with the run time and static resources. This specification requires at minimum version 2.1 of the Java Servlet API.

Oracle Event Processing supports the following features for Jetty:

- Section , "Servlets"
- Section , "Network I/O Integration"
- Section , "Thread Pool Integration"
- Section , "Jetty Work Managers"

For details about configuring Jetty, see Section , "Configuring a Jetty Server Instance."

Servlets

In addition to supporting typical (synchronous) Java servlets, Oracle Event Processing supports asynchronous servlets. An asynchronous servlet receives a request, gets a thread and performs some work, and finally releases the thread while waiting for those actions to complete before re-acquiring another thread and sending a response.

Network I/O Integration

Oracle Event Processing uses network I/O (NetIO) to configure the port and listen address of Jetty services.

Note: Jetty has a built-in capability for multiplexed network I/O. However, it does not support multiple protocols on the same port.

Thread Pool Integration

Oracle Event Processing Jetty services use the Oracle Event Processing Work Manager to provide for scalable thread pooling. See Section , "Example Jetty Configuration."

Note: Jetty provides its own thread pooling capability. However, Oracle recommends using the Oracle Event Processing self-tuning thread pool to minimize footprint and configuration complexity.

Jetty Work Managers

Oracle Event Processing allows you to configure how your application prioritizes the execution of its work. Based on rules you define and by monitoring actual run time performance, you can optimize the performance of your application and maintain service level agreements. You define the rules and constraints for your application by defining a work manager.

This section describes:

- Section , "Understanding How Oracle Event Processing Uses Thread Pools"
- Section , "Understanding Work Manager Configuration"

For more information, see Section , "work-manager Configuration Object."

Understanding How Oracle Event Processing Uses Thread Pools

Oracle Event Processing uses is a single thread pool, in which all types of work are executed. Oracle Event Processing prioritizes work based on rules you define, and run-time metrics, including the actual time it takes to execute a request and the rate at which requests are entering and leaving the pool.

The common thread pool changes its size automatically to maximize throughput. The queue monitors throughput over time and based on history, determines whether to adjust the thread count. For example, if historical throughput statistics indicate that a higher thread count increased throughput, Oracle Event Processing increases the thread count. Similarly, if statistics indicate that fewer threads did not reduce throughput, Oracle Event Processing decreases the thread count.

Understanding Work Manager Configuration

Oracle Event Processing prioritizes work and allocates threads based on an execution model that takes into account defined parameters and run-time performance and throughput.

You can configure a set of scheduling guidelines and associate them with one or more applications, or with particular application components. For example, you can associate one set of scheduling guidelines for one application, and another set of guidelines for other applications. At run time, Oracle Event Processing uses these guidelines to assign pending work and enqueued requests to execution threads.

To manage work in your applications, you define one or more of the following work manager components:

- `fairshare`—Specifies the average thread-use time required to process requests.

For example, assume that Oracle Event Processing is running two modules. The Work Manager for `ModuleA` specifies a `fairshare` of 80 and the Work Manager for `ModuleB` specifies a `fairshare` of 20.

During a period of sufficient demand, with a steady stream of requests for each module such that the number requests exceed the number of threads, Oracle Event Processing allocates 80% and 20% of the thread-usage time to `ModuleA` and `ModuleB`, respectively.

> **Note:** The value of a fair share request class is specified as a relative value, not a percentage. Therefore, in the above example, if the request classes were defined as 400 and 100, they would still have the same relative values.

- `max-threads-constraint`—This constraint limits the number of concurrent threads executing requests from the constrained work set. The default is unlimited. For example, consider a constraint defined with maximum threads of 10 and shared by 3 entry points. The scheduling logic ensures that not more than 10 threads are executing requests from the three entry points combined.

  A `max-threads-constraint` can be defined in terms of a the availability of resource that requests depend upon, such as a connection pool.

  A `max-threads-constraint` might, but does not necessarily, prevent a request class from taking its fair share of threads or meeting its response time goal. Once the constraint is reached the Oracle Event Processing does not schedule requests of this type until the number of concurrent executions falls below the limit. The Oracle Event Processing then schedules work based on the fair share or response time goal.

- `min-threads-constraint`—This constraint guarantees a number of threads the server will allocate to affected requests to avoid deadlocks. The default is zero. A `min-threads-constraint` value of one is useful, for example, for a replication update request, which is called synchronously from a peer.

  A `min-threads-constraint` might not necessarily increase a fair share. This type of constraint has an effect primarily when the Oracle Event Processing instance is close to a deadlock condition. In that case, it the constraint causes Oracle Event Processing to schedule a request even if requests in the service class have gotten more than their fair share recently.

Configuring a Jetty Server Instance

You use the following configuration objects to configure an instance of the Jetty HTTP server in the `config.xml` file that describes your Oracle Event Processing domain:

- `jetty`: See Section , "jetty Configuration Object" for details.
- `netio`: See Section , "netio Configuration Object" for details.
- `work-manager`: See Section , "work-manager Configuration Object" for details.
- `jetty-web-app`: See Section , "jetty-web-app Configuration Object" for details

For information on security configuration tasks that affect Jetty, see Section , "Configuring Jetty Security".

For more information, see:

- Section , "Developing Servlets for Jetty"

- Section , "Web Application Deployment"
- Section , "Example Jetty Configuration"

jetty Configuration Object

Use the parameters described in the following table to define a `jetty` configuration object in your `config.xml` file.

Table 11–1 Configuration Parameters for the jetty Element

Parameter	Type	Description
`network-io-name`	`String`	The name of the NetIO service used. The NetIO service defines the port the server listens on. See Section , "netio Configuration Object" for details.
`work-manager-name`	`String`	The name of the Work Manager that should be used for thread pooling. If not specified, the default work manager is used. See Section , "work-manager Configuration Object."
`scratch-directory`	`String`	The name of a directory where temporary files required for web applications, JSPs, and other types of Web artifacts are kept.
`debug-enabled`	`boolean`	Enable debugging in the Jetty code using the OSGi Log Service.
`name`	`String`	The name of the jetty server instance.

netio Configuration Object

Use the parameters described in the following table to define a `netio` configuration object in your `config.xml` file.

Table 11–2 Configuration Parameters for the netio Element

Parameter	Type	Description
`name`	`String`	The name of this configuration object.
`port`	`int`	The listening port number.
`listen-address`	`String`	The address on which an instance of netio service listens for incoming connections. ▪ It may be set to a numeric IP address in the a.b.c.d format, or to a host name. ▪ If not set, the service listens on all network interfaces. The value of this parameter cannot be validated until the service has started.

work-manager Configuration Object

Use the parameters described in the following table to define a `work-manager` configuration object in your `config.xml` file.

Table 11–3 Configuration Parameters for the work-manager Element

Parameter	Type	Description
`min-threads-constraint`	`Integer`	The minimum threads this work manager uses.
`fairshare`	`Integer`	The fairshare value this work manager uses.
`max-threads-constraint`	`Integer`	The maximum threads constraint this work manager uses.
`name`	`String`	The name of this work manager.

jetty-web-app Configuration Object

Use the following configuration object to define a Web application for use by Jetty:

Table 11–4 Configuration Parameters for the jetty-web-app Element

Parameter	Type	Description
`context-path`	`String`	The context path where this web app is deployed in the web server's name space. If not set, it defaults to "`/`".
`scratch-directory`	`String`	The location where Jetty stores temporary files for this web app. Overrides the `scratch-directory` parameter in the Section , "Configuring a Jetty Server Instance."
`path`	`String`	A file name that points to the location of the web app on the server. It may be a directory or a WAR file.
`jetty-name`	`String`	The name of the Jetty service where this web application is deployed. It must match the name of an existing Section , "Configuring a Jetty Server Instance."
`name`	`String`	The name of this configuration object.

Developing Servlets for Jetty

Oracle Event Processing supports development of servlets for deployment to Jetty by creating a standard Java EE Web Application and configuring it using the Section , "jetty-web-app Configuration Object."

Web Application Deployment

Oracle Event Processing supports deployments packaged either as WAR files or as exploded WAR files, as described in version 2.4 of the Java Servlet Specification.

You can deploy pre-configured web applications from an exploded directory or WAR file by including them in the server configuration.

Security constraints specified in the standard `web.xml` file are mapped to the Common Security Services security provider. The Servlet API specifies declarative role-based security, which means that particular URL patterns can be mapped to security roles.

Example Jetty Configuration

The following snippet of a `config.xml` file provides an example Jetty configuration; only Jetty-related configuration information is shown:

Example 11–1 Example Jetty Configuration

```
<config>
  <netio>
    <name>JettyNetIO</name>
    <port>9002</port>
  </netio>
  <work-manager>
    <name>WM</name>
    <max-threads-constraint>64</max-threads-constraint>
    <min-threads-constraint>3</min-threads-constraint>
  </work-manager>
  <jetty>
    <name>TestJetty</name>
    <work-manager-name>WM</work-manager-name>
    <network-io-name>JettyNetIO</network-io-name>
    <debug-enabled>false</debug-enabled>
    <scratch-directory>JettyWork</scratch-directory>
  </jetty>
  <jetty-web-app>
    <name>test</name>
```

```
    <context-path>/test</context-path>
    <path>testWebApp.war</path>
    <jetty-name>TestJetty</jetty-name>
  </jetty-web-app>
</config>
```

12

Configuring JMX for Oracle Event Processing

This chapter describes how to configure Java Management Extensions (JMX) for use with Oracle Event Processing, including accessing MBeans, JMX configuration objects and programmatically connecting and configuring components with JMX APIs.

This chapter includes the following sections:

Overview of JMX Support in Oracle Event Processing

Oracle Event Processing provides standards-based interfaces that are fully compliant with the Java Management Extensions (JMX) specification. Software developers can use these interfaces to monitor Oracle Event Processing MBeans, to change the configuration of an Oracle Event Processing domain, and to monitor Oracle Event Processing applications.

For more information, see:

- Section , "Understanding JMX Configuration"
- Section , "Understanding JMX Management"

Understanding JMX Configuration

Before you can manage Oracle Event Processing applications, servers, and domains using JMX and Oracle Event Processing MBeans, you must first configure the JMX service on your Oracle Event Processing server.

You configure the Oracle Event Processing JMX service using the following elements in the `config.xml` file that describes your Oracle Event Processing domain:

- `jmx`: See Section , "jmx Configuration Object" for details.
- `rmi`: See Section , "rmi Configuration Object" for details.
- `jndi-context`: See Section , "jndi-context Configuration Object" for details.
- `exported-jndi-context`: See Section , "exported-jndi-context Configuration Object" for details

For more information, see Section , "Configuring JMX".

Understanding JMX Management

Oracle Event Processing applications define an event processing network (EPN) that is made up of components such as adapters, channels, and processors. You deploy these applications to an Oracle Event Processing instance that has been started in a domain.

> **Note:** Components are also sometimes referred to as stages, in particular in the management Javadocs. However, for consistency with the rest of the Oracle Event Processing documentation, this section uses the term components.

You can dynamically configure each component in the EPN using managed beans, or *MBeans*. Typical configuration tasks include adding and removing Oracle CQL or EPL rules, changing channel max size, subscribing to notifications, and executing operations.

You manipulate the MBeans using any of the following:

- Oracle Event Processing Visualizer: the Oracle Event Processing graphical administration console.

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

- `wlevs.Admin`: the Oracle Event Processing command-line administration utility.

  For more information, see Appendix A, "wlevs.Admin Command-Line Reference".

- Deployer: the Oracle Event Processing command-line deployment utility.

  For more information, see Appendix B, "Deployer Command-Line Reference".

- `jconsole`: the JMX console that the Java JDK provides.

- Your own Java code using standard JMX APIs: `http://java.sun.com/javase/technologies/core/mntr-mgmt/javamanagement`.

This section describes:

- Section , "Accessing the Oracle Event Processing JMX Server"
- Section , "Accessing Configuration MBeans"
- Section , "Accessing Oracle Event Processing Runtime MBeans"

For more information, see:

- Section , "Understanding Oracle Event Processing MBeans"
- Section , "Managing With JMX"

Accessing the Oracle Event Processing JMX Server

To access Oracle Event Processing MBeans, you must first connect to the Oracle Event Processing JMX server.

Oracle Event Processing does not support the JRMP protocol. Instead, JMX clients must use the more secure MSA protocol for both local and remote access to the Oracle Event Processing JMX server.

When you connect to the Oracle Event Processing JMX server that is running on `localhost` or on a remote host, you must copy the following Oracle Event Processing server JAR files to the client classpath of the host from which you want to connect to the Oracle Event Processing server:

- *ORACLE_CEP_HOME*\modules\com.bea.core.jmx_8.0.0.0.jar
- *ORACLE_CEP_HOME*\modules\com.bea.core.rmi_7.0.0.0.jar
- *ORACLE_CEP_HOME*\modules\com.bea.core.jndi.context_8.0.0.0.jar
- *ORACLE_CEP_HOME*\modules\com.bea.core.logging_1.8.0.0.jar
- *ORACLE_CEP_HOME*\modules\com.bea.core.bootbundle_11.0.0.0.jar

Where *ORACLE_CEP_HOME* refers to the directory in which you installed Oracle Event Processing (such as `d:\oracle_home`).

You must launch your JMX client (such as `jconsole`) using the following command line options and classpath (split for readability; in practice, the command should be on one line):

```
prompt> java -Djmx.remote.protocol.provider.pkgs=com.bea.core.jmx.remote.provider
-Dmx4j.remote.resolver.pkgs=com.bea.core.jmx.remote.resolver
-Djava.naming.factory.initial=com.bea.core.jndi.context.ContextFactory
-classpath %JAVA_HOME%\lib\jconsole.jar;MODULE_HOME\modules\com.bea.core.jmx_
8.0.0.0.jar;
MODULE_HOME\modules\com.bea.core.rmi_7.0.0.0.jar;MODULE_
HOME\modules\com.bea.core.jndi.context_7.0.0.0.jar;
MODULE_HOME\modules\com.bea.core.logging_1.5.0.0.jar;MODULE_
HOME\modules\com.bea.core.bootbundle_8.0.0.0.jar
sun.tools.jconsole.JConsole
```

Where *MODULE_HOME* is the directory you copied the Oracle Event Processing server JAR files to.

To connect to the Oracle Event Processing JMX server, you must use the JMX URL `service:jmx:msarmi://`*HOST-NAME*`:`*port*`/jndi/jmxconnector` so that you are always using the MSA connector (where *HOST-NAME* is either `localhost` or the name of the remote host and *port* is the Oracle Event Processing server JNDI port).

For more information, see:

- Section , "How to Programmatically Connect to the Oracle Event Processing JMX Server From a Non-Oracle Event Processing Client"
- Section , "How to Connect to a Local or Remote Oracle Event Processing JMX Server Using JConsole With Security Disabled"

Accessing Configuration MBeans

You can also perform some configuration and application life cycle management of the server, domain, and deployed applications using MBeans, although this section predominantly describes configuring individual application components. However, because server, domain, and application configuration is also done using MBeans, much of the information in this section is applicable.

Each component in a deployed application (adapter, channel, or processor) has a *configuration MBean* that manages the underlying configuration of the component. Each type of component has its own set of manageable artifacts. For example, you can dynamically configure the maximum number of threads for a channel or the Oracle CQL rules associated with a processor.

Accessing Oracle Event Processing Runtime MBeans

You can also gather monitoring information for each component in the EPN using *runtime MBeans*. Monitoring information includes throughput (number of events

passing through a component) and latency (how long it takes an event to pass through a component).

Understanding Oracle Event Processing MBeans

Oracle Event Processing exposes the following types of MBeans:

- **Configuration MBeans**—Contain information about the configuration of components in an EPN, a deployed Oracle Event Processing application, the server and domain configurations. These MBeans have a fixed management interface and represent the information contained in the domain config.xml file and the component configuration XML files. Examples of standard MBeans include `CQLProcessorMBean` and `EventChannelMBean`.
- **Runtime MBeans**—Contain information about throughput and latency of a component.

For full reference information about Oracle Event Processing MBeans and management in general, see the following classes in the *Oracle Fusion Middleware Java API Reference for Oracle Event Processing*:

- `com.bea.wlevs.management.configuration`
- `com.bea.wlevs.management.runtime`
- `com.bea.wlevs.monitor.management`
- `com.bea.wlevs.monitor`
- `com.bea.wlevs.processor.epl.management`
- `com.bea.wlevs.deployment.mbean`

This section describes:

- Section , "Oracle Event Processing Configuration MBeans"
- Section , "Oracle Event Processing Runtime MBeans"
- Section , "Oracle Event Processing MBean Hierarchy"

Oracle Event Processing Configuration MBeans

When you deploy an Oracle Event Processing application, the server automatically creates a configuration MBean for each component in the EPN whose manageability has been enabled, or in other words, for each component registered in the EPN assembly file. If you have extended the configuration of an adapter, then the server deploys a custom configuration MBean for the adapter.

Using JMX, you can dynamically configure the component using its configuration MBean. For example, using the `StreamMBean.setMaxSize()` method you can set the size of a channel component.

Configuration MBean Naming Oracle Event Processing configuration MBeans are arranged in a hierarchy. The object name of each MBean reflects its position in the hierarchy. A typical object naming pattern is as follows:

`com.bea.wlevs:Name=`*`name`*`,Type=`*`type`*`,[`*`TypeOfParentMBean=NameOfParentMBean`*`]`

where:

- `com.bea.wlevs:` is the JMX domain name.
- `Name=`*`name`*`,Type=`*`type`*`,[`*`TypeOfParentMBean=NameOfParentMBean`*`]` is a set of JMX key properties.

The order of the key properties is not significant, but the object name must begin with `com.bea:wlevs:`.

For example, the object name of the MBean corresponding to a processor called `myprocessor` in the application `myapplication` in the domain is as follows:

```
com.bea.wlevs:Name=myprocessor,Type=EPLProcessor,Application=myapplication
```

Table 12–1 describes the key properties that Oracle Event Processing encodes in its MBean object names.

Table 12–1 Oracle Event Processing MBean Object Name Key Properties

This Key Property	Specifies
`Name=`*`name`*	The string that you provided when you created the resource that the MBean represents. This is typically the name of a component. The name of a particular component is specified in the EPN assembly file using the `id` attribute of the component registration. For example, in the case of processors, the entry in the EPN assembly file might look like the following: `<wlevs:processor id="myprocessor" advertise="true" />` In this case, the key property would be `Name=myprocessor`.
`Type=`*`type`*	The short name of the MBean's type. The short name is the unqualified type name without the `MBean` suffix. For example, for an MBean that is an instance of the `CQLProcessorMBean`, use `CQLProcessor`. In this case, the key property would be `Type=CQLProcessor`.

`TypeOfParentMBean=`*`NameOfParentMBean`*	Specifies the type and name of the parent MBean.
	For components, this is always `Application=`*`application_name`*, where *`application_name`* refers to the name of the application of which the component is a part.
	The name of a particular Oracle Event Processing application is specified with the `Bundle-SymbolicName` header of the `MANIFEST.MF` file of the application bundle. For example, if an application has the following `MANIFEST.MF` snippet (only relevant parts are shown):
	`Manifest-Version: 1.0`
	`Archiver-Version:`
	`Build-Jdk: 1.5.0_06`
	`....`
	`Bundle-SymbolicName: myapplication`
	then the key property would be `Application=myapplication`.

Table 12–2 shows examples of configuration MBean objects names that correspond to the component declarations in the HelloWorld sample EPN assembly file. In each example, the application name is `helloworld` and the domain name is `mydomain`.

Table 12–2 Component Declaration Example With Corresponding MBean Object Names

EPN Assembly File Component Declaration	Corresponding Configuration MBean Object Name
`<wlevs:processor id="helloworldProcessor" />`	`com.bea.wlevs:Name=helloworldProcessor,Type=EPLProcessor,Application=helloworld,Domain=mydomain` `EPLProcessor` is the standard configuration MBean for processor components. The manageable property is `rules`.

Table 12–2 (Cont.) Component Declaration Example With Corresponding MBean Object

EPN Assembly File Component Declaration	Corresponding Configuration MBean Object Name

`<wlevs:channel id="helloworldInstream">`
`  <wlevs:listener ref="helloworldProcessor"/>`
`  <wlevs:source ref="helloworldAdapter"/>`
`</wlevs:channel>`
	`com.bea.wlevs:Name=helloworldInstream,Type=Channel,Application=helloworld,Domain=mydomain`
	`Channel` is the standard configuration MBean for a channel component. The manageable properties are `MaxSize` and `MaxThreads`.

Oracle Event Processing Runtime MBeans

You can also gather monitoring information for each component in the EPN using *runtime MBeans*. Oracle Event Processing server defines the following metrics that you can monitor for each component:

- Throughput—The number of events processed by the component. The parameters for this metric are: throughput time interval, aggregation time interval, the unit of time for the intervals.
- Average Latency—The average amount of time it takes an event to pass through a component, or *latency*. Parameters: aggregation time interval, the unit of time for the interval.
- Maximum Latency—The maximum amount of time it takes an event to pass through a component. Parameters: aggregation time interval, the unit of time for the interval.
- Average Latency Threshold—Specifies whether the average latency of events between the start- and end-points of a component crosses a specified threshold. Parameters: aggregation time interval, threshold, the unit of time for the interval.

Runtime MBean Naming Runtime MBeans are named using the same pattern as with configuration mbeans except for one extra property: `Direction`. This property has two valid values: `OUTBOUND` or `INBOUND` that refer to the point at which you want to gather the statistic `OUTBOUND` means that you want to gather throughput or latency as events flow out of the specified component; similarly `INBOUND` means you want to gather the monitoring information as events flow into a component.

For example, the object name of the runtime MBean corresponding to a processor called `myprocessor` in the application `myapplication`, in which events will be monitored as they flow into the component, is as follows:

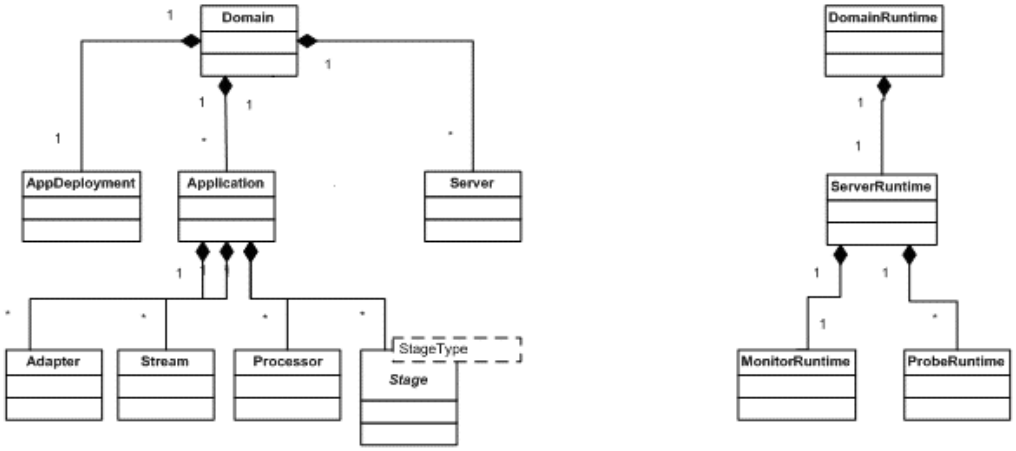
```
com.bea.wlevs:Name=myprocessor,Type=EPLProcessor,Application=myapplication,Direction=INBOUND
```

See Section , "Configuration MBean Naming" for details about configuration MBean naming.

Oracle Event Processing MBean Hierarchy

Figure 12–1 describes the Oracle Event Processing MBean tree.

Figure 12–1 Oracle Event Processing MBean Tree



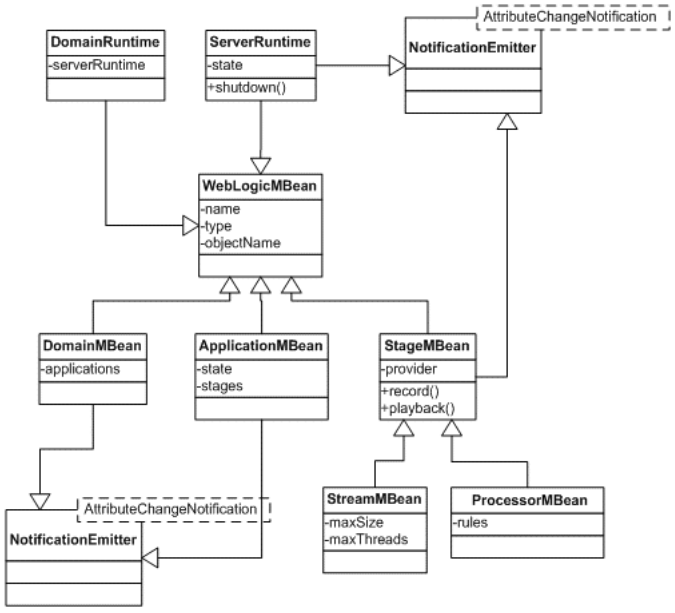
All MBeans must be registered in an MBean server under an object name of type `javax.management.ObjectName`. Oracle Event Processing follows a convention in which object names for child MBeans contain part of its parent MBean object name.

There are two main MBean roots: `DomainMBean` and `DomainRuntimeMBean`. The former includes configuration MBeans for the entire domain, the latter contains runtime information, such as statistics, and local services, such as Monitor, that are generally scoped to a single server instance.

`ApplicationMBean` is a child of the `DomainMBean` instead of the `ServerMBean`. This is because an application is unique within a domain, and can span multiple servers.

Figure 12–2 shows the main classes and relationships that make up the object model.

Figure 12–2 Oracle Event Processing MBean Object Model



Most MBeans are notification emitters that generate `AttributeChangeNotifications`. In other words, a JMX client can register to receive attribute change notifications regarding changes to application state, insertion and removal of applications at the domain, channel size and thread changes, insertion and removal of rules, and so on.

Configuring JMX

You configure the Oracle Event Processing JMX service using the following elements in the `config.xml` file that describes your Oracle Event Processing domain:

- `jmx`: See Section , "jmx Configuration Object" for details.
- `rmi`: See Section , "rmi Configuration Object" for details.
- `jndi-context`: See Section , "jndi-context Configuration Object" for details.
- `exported-jndi-context`: See Section , "exported-jndi-context Configuration Object" for details

For information on security configuration tasks that affect JMX, see Section , "Configuring JMX Security".

For more information, see:

- Section , "Understanding JMX Configuration"
- Section , "Example of Configuring JMX"

jmx Configuration Object

Table 12–3 lists the `jmx` element child elements in the `config.xml` file that you must configure.

Table 12–3 Configuration Parameters for the jmx Element

Parameter	Type	Description
`rmi-service-name`	`String`	The name of the RMI service with which the jmx server will register to receive calls.
`jndi-service-name`	`String`	The name of the JNDI service to which the jmx server will bind its object.

rmi Configuration Object

The Oracle Event Processing RMI service provides:

- Ability to register a POJO interface in a server for remote method invocation from a client.
- Ability to register for any context propagation from the client to the server on a remote method invocation, intercept, and act on this propagated context in the server.

Table 12–4 lists the `rmi` element child elements in the `config.xml` file that you use to export server-side objects to remote clients.

Table 12–4 Configuration Parameters for the rmi Element

Parameter	Type	Description
`heartbeat-period`	`int`	The number of failed heartbeat attempts before triggering disconnect notifications to all registered listeners.
`http-service-name`	`String`	The name of the HTTP service used to register remote objects (such as Jetty, see Section , "Overview of Jetty Support in Oracle Event Processing").
`heartbeat-interval`	`int`	The amount of time, in milliseconds, between heartbeats. Once the number of unsuccessful heartbeat attempts has reached the value specified by the `HeartbeatPeriod` parameter, all registered `DisconnectListener` instances are notified.
`name`	`String`	The name of this configuration object.

jndi-context Configuration Object

The JNDI Factory Manager is responsible for supporting JNDI in an OSGi environment. It allows JNDI providers to be supplied as OSGi bundles, and for code running inside OSGi bundles to have full access to the JNDI environment.

The Factory Manager consists of two components:

- An OSGi bundle, which provides the OSGi-specific factory management code, to look up JNDI objects using the appropriate OSGi classloader.
- JNDI "glue code," internal to Oracle Event Processing, that initializes the JNDI environment to support the factory manager bundle.

Table 12–5 lists the `jndi-context` element child elements in the `config.xml` file that you must configure.

Table 12–5 Configuration Parameters for the jndi-context Element

Parameter	Type	Description
`default-provider`	`boolean`	If `true`, the default Oracle Event Processing JNDI provider is used. Default value is `true`.
`name`	`String`	The name of this configuration object.

exported-jndi-context Configuration Object

Requires a configured Section , "jndi-context Configuration Object."

Use this configuration object to export a remote JNDI service to a client using RMI. A JNDI context is registered with the RMI service to provide remote access to clients that pass a provider URL parameter in their `InitialContext` object.

Table 12–6 lists the `exported-jndi-context` element child elements in the `config.xml` file that you must configure.

Table 12–6 Configuration Parameters for the exported-jndi-context Element

Parameter	Type	Description
`rmi-service-name`	`String`	The name of the RMI service that should be used to serve this JNDI context over the network. It must match an existing `<rmi>` configuration object. See Section , "rmi Configuration Object."
`name`	`String`	The name of this configuration object. The value of this element must be different from the value of the `<name>` child element of `<jndi-context>` in the same `config.xml` file.

Example of Configuring JMX

Example 12–1 shows a `config.xml` snippet with JMX configuration; only relevant parts of the file are shown.

Example 12–1 JMX Configuration

```
<config>
  <netio>
    <name>JettyNetio</name>
    <port>12345</port>
  </netio>
  <work-manager>
    <name>WM</name>
    <fairshare>5</fairshare>
    <min-threads-constraint>1</min-threads-constraint>
    <max-threads-constraint>4</max-threads-constraint>
```

```
  </work-manager>
  <jetty>
    <name>TestJetty</name>
    <work-manager-name>WM</work-manager-name>
    <network-io-name>JettyNetio</network-io-name>
  </jetty>
  <rmi>
    <name>RMI</name>
    <http-service-name>TestJetty</http-service-name>
  </rmi>
  <jndi-context>
    <name>JNDI</name>
  </jndi-context>
  <exported-jndi-context>
    <name>exportedJNDI</name>
    <rmi-service-name>RMI</rmi-service-name>
  </exported-jndi-context>
  <jmx>
    <jndi-service-name>JNDI</jndi-service-name>
    <rmi-service-name>RMI</rmi-service-name>
  </jmx>
</config>
```

Managing With JMX

This section describes detailed examples of managing Oracle Event Processing components using JMX, including:

- Section , "How to Programmatically Connect to the Oracle Event Processing JMX Server From a Non-Oracle Event Processing Client"
- Section , "How to Programmatically Connect to the Oracle Event Processing JMX Server From an Oracle Event Processing Client"
- Section , "How to Programmatically Configure an Oracle Event Processing Component Using JMX APIs"
- Section , "How to Programmatically Monitor the Throughput and Latency of an Oracle Event Processing Component Using JMX APIs"
- Section , "How to Connect to a Local or Remote Oracle Event Processing JMX Server Using JConsole With Security Enabled"
- Section , "How to Connect to a Local or Remote Oracle Event Processing JMX Server Using JConsole With Security Disabled"

> **Note:** When using JConsole, you must start it with the Oracle Event Processing `wlevsjconsole.cmd` or `wlevsjconsole.sh` script. You cannot start `jconsole` directly.

For more information, see:

- Section , "Understanding JMX Configuration"
- Section , "Understanding JMX Management"

How to Programmatically Connect to the Oracle Event Processing JMX Server From a Non-Oracle Event Processing Client

This section describes how to write Java code using the JMX API (`http://java.sun.com/javase/technologies/core/mntr-mgmt/javamanagement`) to

connect to the Oracle Event Processing JMX server from a non-Oracle Event Processing client. This is the first step to all programmatic JMX management.

For information on connecting to the Oracle Event Processing JMX server from another Oracle Event Processing server, see Section , "How to Programmatically Connect to the Oracle Event Processing JMX Server From an Oracle Event Processing Client".

To programmatically connect to the Oracle Event Processing JMX server from a non-Oracle Event Processing client:

1. Be sure that the JMX service is configured for your domain.

   For details see Section , "Configuring JMX".

2. Write the `http://java.sun.com/javase/technologies/core/mntr-mgmt/javamanagement` Java code to configure the component using the appropriate MBean.

   Consider the following JMX programming hints.

   One of the first things you must do in your JMX program is to establish a connection to the JMX server running in the Oracle Event Processing server as Example 12–2 shows.

Example 12–2 Establishing a Connection to the Oracle Event Processing JMX Server

```
public static void initConnection(String hostname, int port, String username, char[]
password)
  throws IOException,MalformedURLException {

  Map<String,Object> env = makeSecureEnv();
  env.put("jmx.remote.protocol.provider.pkgs","com.bea.core.jmx.remote.provider");
  env.put("mx4j.remote.resolver.pkgs","com.bea.core.jmx.remote.resolver");
  env.put("java.naming.factory.initial","com.bea.core.jndi.context.ContextFactory");

  JMXServiceURL serviceUrl = new JMXServiceURL(
      "MSARMI","localhost",9002,"/jndi/jmxconnector"
  );

  System.out.println("Service: " + serviceURL.toString());

  JMXConnector connector = JMXConnectorFactory.connect(serviceUrl,env);

  MBeanServerConnection connection = connector.getMBeanServerConnection();
}

// The JMXConnectorFactory.connect() method's second parameter is a Map object that sets up a
// secure environment using the makeSecureEnv() method, which looks like the following:

private static Map<String,Object> makeSecureEnv() {
    Map<String,Object> env = new HashMap<String,Object>();
    String username = "wlevs" ;
    char[] password = { 'w','l','e','v','s' };
    env.put(JMXConnector.CREDENTIALS, new Serializable[]{username,password});
    env.put("jmx.remote.authenticator", "com.bea.core.jmx.server.CEAuthenticator");
    System.setProperty("jmx.remote.authenticator",
"com.bea.core.jmx.server.CEAuthenticator");
    return env;
}
```

How to Programmatically Connect to the Oracle Event Processing JMX Server From an Oracle Event Processing Client

This section describes how to write Java code using the JMX API (http://java.sun.com/javase/technologies/core/mntr-mgmt/javamanagement) to connect to the Oracle Event Processing JMX server from another Oracle Event Processing server. This is the first step to all programmatic JMX management.

For information on connecting to the Oracle Event Processing JMX server from a non-Oracle Event Processing client, see Section , "How to Programmatically Connect to the Oracle Event Processing JMX Server From a Non-Oracle Event Processing Client".

To programmatically connect to the Oracle Event Processing JMX server from an Oracle Event Processing client:

1. Be sure that the JMX service is configured for your domain.

   For details see Section , "Configuring JMX".

2. Write the http://java.sun.com/javase/technologies/core/mntr-mgmt/javamanagement Java code to configure the component using the appropriate MBean.

   Consider the following JMX programming hints.

   One of the first things you must do in your JMX program is to establish a connection to the JMX server running in the Oracle Event Processing server as Example 12–2 shows.

Example 12–3 Establishing a Connection to the Oracle Event Processing JMX Server

```
public static void initConnection(String hostname, int port, String username, char[]
password)
  throws IOException,MalformedURLException {

  Map<String,Object> env = makeSecureEnv();

  // This is an OSGi necessity
  env.put(
      JMXConnectorFactory.DEFAULT_CLASS_LOADER,
      com.bea.core.jmx.remote.provider.msarmi.ServerProvider.class.getClassLoader()
  );
  env.put(
      JMXConnectorFactory.PROTOCOL_PROVIDER_CLASS_LOADER,
      com.bea.core.jmx.remote.provider.msarmi.ServerProvider.class.getClassLoader()
  );

  JMXServiceURL serviceUrl = new JMXServiceURL(
      "MSARMI","localhost",9002,"/jndi/jmxconnector"
  );

  System.out.println("Service: " + serviceURL.toString());

  env.put(
      JMXConnectorFactory.PROTOCOL_PROVIDER_PACKAGES,
      "com.bea.core.jmx.remote.provider"
  );

  System.setProperty("mx4j.remote.resolver.pkgs", "com.bea.core.jmx.remote.resolver");

  JMXConnector connector = JMXConnectorFactory.connect(url, env);
  connector.connect();

  MBeanServerConnection connection = connector.getMBeanServerConnection();
  ...
```

```
}

// The JMXConnectorFactory.connect() method's second parameter is a Map object that sets up a
// secure environment using the makeSecureEnv() method, which looks like the following:

private static Map<String,Object> makeSecureEnv() {
    Map<String,Object> env = new HashMap<String,Object>();
    String username = "wlevs" ;
    char[] password = { 'w','l','e','v','s' };
    env.put(JMXConnector.CREDENTIALS, new Serializable[]{username,password});
    env.put("jmx.remote.authenticator", "com.bea.core.jmx.server.CEAuthenticator");
    System.setProperty("jmx.remote.authenticator",
"com.bea.core.jmx.server.CEAuthenticator");
    return env;
}
```

How to Programmatically Configure an Oracle Event Processing Component Using JMX APIs

This section describes how to write Java code using the JMX API (http://java.sun.com/javase/technologies/core/mntr-mgmt/javamanagement) to access Oracle Event Processing MBeans.

To programmatically configure an Oracle Event Processing component using JMX APIs:

1. Acquire a connection to the Oracle Event Processing JMX server.

   For details see Section , "How to Programmatically Connect to the Oracle Event Processing JMX Server From a Non-Oracle Event Processing Client".

2. Write the `http://java.sun.com/javase/technologies/core/mntr-mgmt/javamanagement` Java code to configure the component using the appropriate MBean.

   Consider the following JMX programming hints.

   Example 12–4 shows how to use the connection to start getting information about the domain and its deployed applications by querying MBeans.

   First the code shows how to get all MBeans whose type is Domain; there should only be one. Then, using the `DomainMBean`, the sample shows how to retrieve a list of all the deployed applications in the domain (using `ApplicationMBean`):

Example 12–4 Querying MBeans

```
Set domainObjectNames = connection.queryMBeans(
    ObjectName.getInstance(
        ManagementConstants.DOMAIN_NAME + ":" +
        ManagementConstants.TYPE_PROPERTY + "=" +
        DomainMBean.MBEAN_TYPE + ",*"
    ),
    null
);
ObjectName domainName =  ((ObjectInstance)
domainObjectNames.iterator().next()).getObjectName();
System.out.println("Domain Name: " + domainName.getKeyProperty(ManagementConstants.NAME_
PROPERTY));
ObjectName [] applicationNames =
     (ObjectName[]) connection.getAttribute(domainName, "ApplicationMBeans");
ObjectName selectedApplicationObjectName = null ;
for (ObjectName applicationName : applicationNames) {
   String name =
```

```
     applicationName.getKeyProperty(ManagementConstants.NAME_PROPERTY);
   String status =
     (String) connection.getAttribute(applicationName, "State");
    System.out.println("Application: " + name + " Status: " + status);
    selectedApplicationObjectName = applicationName ;
}
```

How to Programmatically Monitor the Throughput and Latency of an Oracle Event Processing Component Using JMX APIs

This section describes how to write Java code using the JMX API (http://java.sun.com/javase/technologies/core/mntr-mgmt/javamanagement) to access Oracle Event Processing MBeans and dynamically monitor the throughput and latency of an Oracle Event Processing component.

To dynamically configure an Oracle Event Processing component using JMX APIs:

1. Acquire a connection to the Oracle Event Processing JMX server.

   For details see Section , "How to Programmatically Connect to the Oracle Event Processing JMX Server From a Non-Oracle Event Processing Client".

2. Acquire an instance of a `MonitorRuntimeMBean` for the component you want to monitor as Example 12–5 shows.

Example 12–5 Acquiring an Instance of a MonitorRuntimeMBean

```
ObjectName processorInbound = ObjectName.getInstance(
    "com.bea.wlevs:Name=myprocessor," +
    "Type=CQLProcessor," +
    "Application=myapplication," +
    "Direction=INBOUND"
);
```

   Be sure you specify whether you want to monitor incoming events (`INBOUND`) or outgoing events (`OUTBOUND`). For example:

3. Use the `MonitorRuntimeMBean` to acquire an instance of `ProbeRuntimeMBean` for the type of statistic you want as Example 12–6 shows.

Example 12–6 Acquiring an Instance of ProbeRuntimeMBean

```
ObjectName monitorName =
    ObjectName.getInstance(
        "com.bea.wlevs:ServerRuntime=localhost," +
        "Name=MonitorRuntime," +
        "Type=MonitorRuntime");

MonitorRuntimeMBean monitorMBean =
    (MonitorRuntimeMBean)MBeanServerInvocationHandler.newProxyInstance(
        connection,
        monitorName,
        MonitorRuntimeMBean.class,
        false);

ObjectName probeName = monitorMBean.monitorAvgThroughput(
    processorInbound,
    1000,
    1000
);

ProbeRuntimeMBean probeOn = (ProveRuntimeMBean)MBeanServerInvocationHandler.newProxyInstance(
    connection,
```

```
    probeName,
    ProbeRuntimeMBean.class,
    false
);
```

The `MonitorRuntimeMBean` has methods for each type of statistic you can gather. For example, you execute `monitorAvgLatency()` if you want to monitor the average latency, `monitorAvgThroughput()` to monitor the average throughput, and so on. These methods all return `ProbeRuntimeMBean`.

4. Use the `ProbeRuntimeMbean` instance to get the actual runtime metrics in one of the following ways:
   a. Use the `ProbeRuntimeMBean` method `getMetric()` to pull the information.
   b. Use `javax.management.NotificationBroadcaster.addNotificationListener()` to have the information pushed to you every time there is a change in the metrics.
5. When you are finished gathering monitoring information, unregister the MBean from the MBean server as Example 12–7 shows.

Example 12–7 Unregistering the MBean

```
probON.terminate();
```

For additional details about these MBean interfaces and how to use them to monitor throughput and latency, see the `com.bea.wlevs.monitor.management` package in the *Oracle Fusion Middleware Java API Reference for Oracle Event Processing*.

How to Connect to a Local or Remote Oracle Event Processing JMX Server Using JConsole With Security Enabled

You can use the `wlevsjconsole` script to connect to an Oracle Event Processing JMX server running on your local host or on a remote host to browse and manage Oracle Event Processing MBeans with the JDK `jconsole`.

This procedure describes how to use JConsole when the Oracle Event Processing server has security enabled. This is the default configuration and is recommended for production servers. Alternatively, you can connect to the JMX server with security disabled (see Section , "How to Connect to a Local or Remote Oracle Event Processing JMX Server Using JConsole With Security Disabled").

For more information, see Section , "Accessing the Oracle Event Processing JMX Server".

> **Note:** When using JConsole, you must start it with the Oracle Event Processing `wlevsjconsole.cmd` or `wlevsjconsole.sh` script. You cannot start `jconsole` directly.

To connect to a local or remote Oracle Event Processing JMX server using JConsole with security enabled:

1. Ensure that the local or remote Oracle Event Processing server is running.
2. Open a command window and set your environment as described in "Setting Your Development Environment" in the *Oracle Fusion Middleware Developer's Guide for Oracle Event Processing for Eclipse*.

3. Launch `jconsole` using the `wlevsjconsole.cmd` or `wlevsjconsole.sh` script located in the *ORACLE_CEP_HOME*`/ocep_11.1/bin` directory, where *ORACLE_CEP_HOME* refers to the directory in which you installed Oracle Event Processing (such as `/oracle_home`).

   a. To connect to a local Oracle Event Processing server, enter:

      ```
      prompt> wlevsjconsole.cmd
      ```

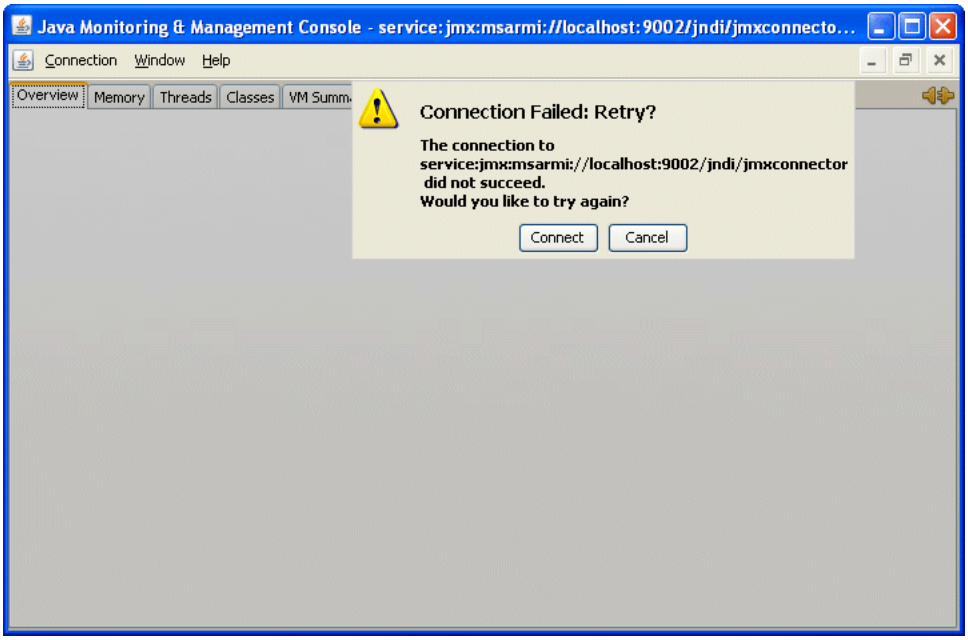
   b. To connect to a remote Oracle Event Processing server, enter:

      ```
      prompt> wlevsjconsole.cmd HOST-NAME:PORT
      ```

      Where *HOST-NAME* is the name of the remote host and *PORT* is the NetIO port as configured in the remote host's *ORACLE_CEP_HOME*`/user_projects/domains/`*DOMAIN-NAME*`/defaultserver/config/config.xml` file.

   The jconsole browser attempts to log into the JMX server and initially fails as Figure 12–3 shows.

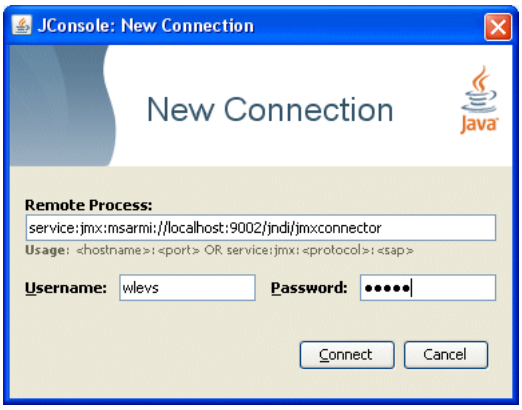
Figure 12–3 Jconsole Initial Login Attempt



4. Click **Cancel**.

   The `Jconsole` New Connection dialog appears as shown in Figure 12–4.

Figure 12–4 JConsole New Connection Dialog



5. Configure the New Connection dialog as Table 12–7 describes.

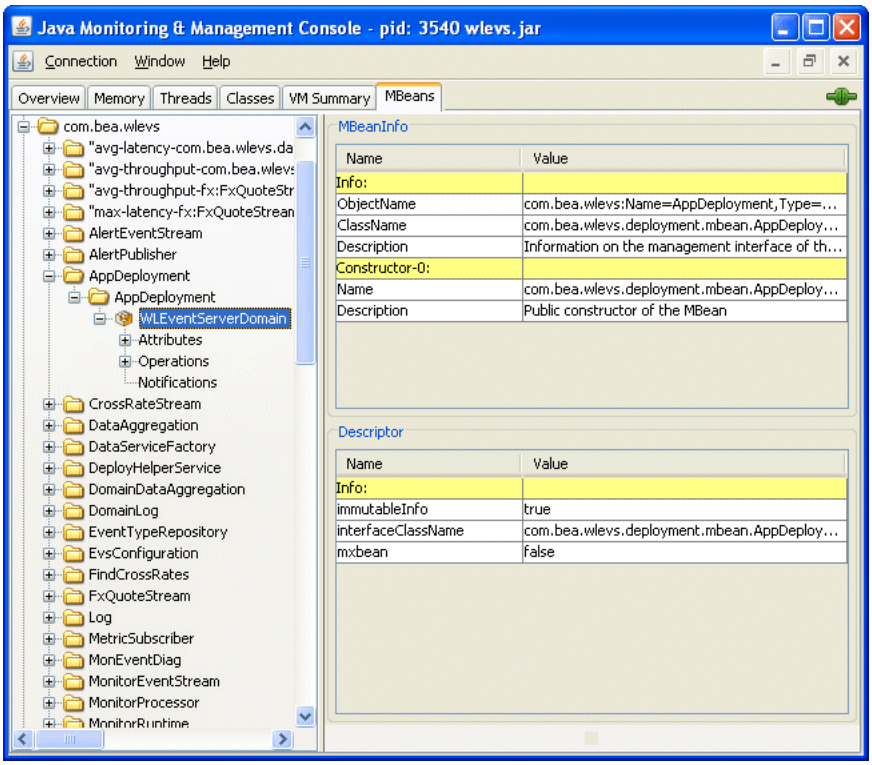
Table 12–7 JConsole New Connection Attributes

Attribute	Description
`Remote Process`	Enter the following URL: `service:jmx:msarmi://`*`HOST-NAME`*`:`*`PORT`*`/jndi/jmxconnector` Where *`HOST-NAME`* is the name of the local or remote host and *`PORT`* is the NetIO port as configured in the remote host's *`ORACLE_CEP_HOME`*`/user_projects/domains/`*`DOMAIN-NAME`*`/defaultserver/config/config.xml` file (default: `9002`).
`Username`	Enter an Oracle Event Processing administration password. Default: **wlevs**.
`Password`	Enter the password for the Oracle Event Processing administration password you chose. Default: **wlevs**.

6. Click **Connect**.

   The `jconsole` browser opens and provides access to Oracle Event Processing MBeans as Figure 12–6 shows.

Figure 12–5 JConsole Browser



How to Connect to a Local or Remote Oracle Event Processing JMX Server Using JConsole With Security Disabled

You can use the `wlevsjconsole` script to connect to an Oracle Event Processing JMX server running on your local host or on a remote host to browse and manage Oracle Event Processing MBeans with the JDK `jconsole`.

This procedure describes how to use JConsole when the Oracle Event Processing server has security disabled. This is a common development configuration and is not recommended for production servers. Alternatively, you can connect to the JMX server with security enabled (see Section , "How to Connect to a Local or Remote Oracle Event Processing JMX Server Using JConsole With Security Enabled").

For more information, see Section , "Accessing the Oracle Event Processing JMX Server".

Note: When using JConsole, you must start it with the Oracle Event Processing `wlevsjconsole.cmd` or `wlevsjconsole.sh` script. You cannot start `jconsole` directly.

To connect to a local or remote Oracle Event Processing JMX server using JConsole with security disabled:

1. Ensure that the local or remote Oracle Event Processing server is running with security disabled.

   For more information, see Section , "Disabling Security".

2. Open a command window and set your environment as described in "Setting Your Development Environment" in the *Oracle Fusion Middleware Developer's Guide for Oracle Event Processing for Eclipse*.

3. Launch `jconsole` using the `wlevsjconsole.cmd` or `wlevsjconsole.sh` script located in the *ORACLE_CEP_HOME*`/ocep_11.1/bin` directory, where *ORACLE_CEP_HOME* refers to the directory in which you installed Oracle Event Processing (such as `/oracle_home`).

   a. To connect to a local Oracle Event Processing server, enter:

   ```
   prompt> wlevsjconsole.cmd
   ```

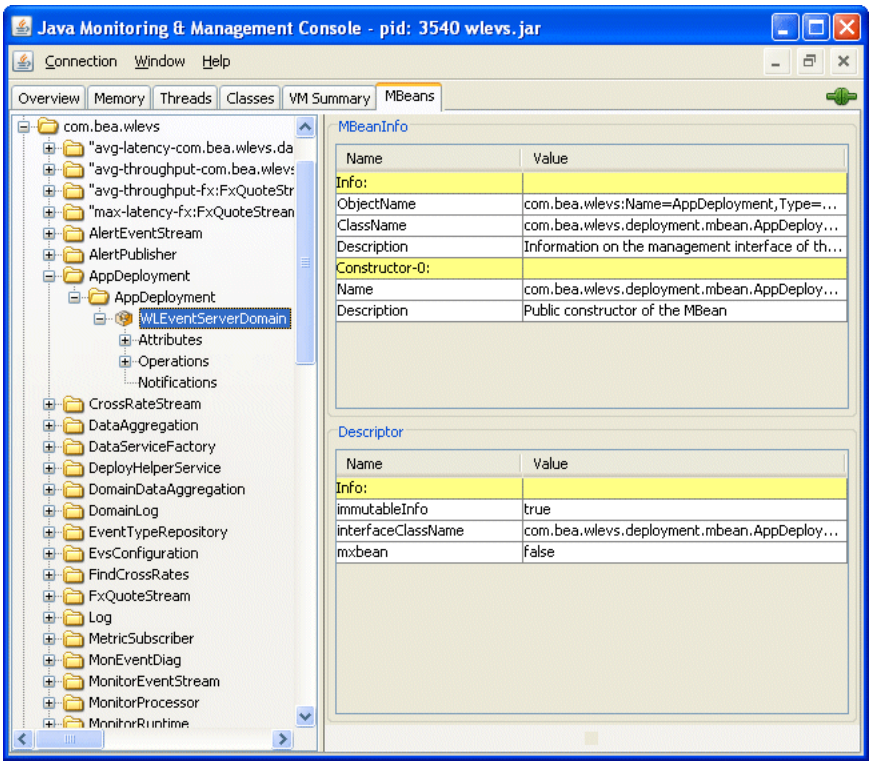
   b. To connect to a remote Oracle Event Processing server, enter:

   ```
   prompt> wlevsjconsole.cmd HOST-NAME:PORT
   ```

   Where *HOST-NAME* is the name of the remote host and *PORT* is the `NetIO` port as configured in the remote host's *ORACLE_CEP_HOME*`/user_projects/domains/`*DOMAIN-NAME*`/defaultserver/config/config.xml` file.

   The script automatically connects to the JMX server and the `jconsole` browser opens and provides access to Oracle Event Processing MBeans as Figure 12–6 shows.

Figure 12–6 JConsole Browser



13

Configuring JDBC for Oracle Event Processing

This chapter describes how to configure Java Database Connectivity (JDBC) for use with Oracle Event Processing, including information on configuring data sources and on using Oracle and Type 4 Data Direct JDBC drivers.

This chapter includes the following sections:

Overview of Database Access from an Oracle Event Processing Application

Oracle Event Processing supports Java Database Connectivity (JDBC) 3.0 (see `http://java.sun.com/products/jdbc/download.html#corespec30`) for relational database access.

The JDBC API (see `http://java.sun.com/javase/technologies/database/index.jsp`) provides a standard, vendor-neutral mechanism for connecting to and interacting with database servers and other types of tabular resources that support the API. The JDBC `javax.sql.DataSource` interface specifies a database connection factory that is implemented by a driver. Instances of `DataSource` objects are used by applications to obtain database connections (instances of `java.sql.Connection`). After obtaining a connection, an application interacts with the resource by sending SQL commands and receiving results.

Oracle Event Processing provides the following JDBC drivers:

- Oracle 11.2 thin driver (see Section , "Oracle JDBC Driver")
- SQL Server Type 4 JDBC Driver from DataDirect (see Section , "Type 4 JDBC Driver for SQL Server from DataDirect")

Optionally, you can use your own JDBC driver (see Section , "How to Access a Database Driver Using bootclasspath").

Oracle Event Processing also provides a `DataSource` abstraction that encapsulates a JDBC driver `DataSource` object and manages a pool of pre-established connections, and the Oracle WebLogic Server WLConnection interface provides useful methods for accessing and manipulating Oracle data sources. For more information, see Section ,

"Description of Oracle Event Processing Data Sources".

Oracle JDBC Driver

Oracle Event Processing includes the Oracle 11.2 Thin driver packaged in the following JAR files:

- *ORACLE_CEP_HOME*/modules/com.bea.oracle.ojdbc5_1.0.0.0_11-2-0-0.jar: for use with Java SE 5.
- *ORACLE_CEP_HOME*/modules/com.bea.oracle.ojdbc6_1.0.0.0_11-2-0-0.jar: for use with Java SE 6.

The JDBC Thin driver is a pure Java, Type IV driver that can be used in applications and applets. It is platform-independent and does not require any additional Oracle software on the client side. The JDBC Thin driver communicates with the server using SQL*Net to access the Oracle Database.

For more information, see:

- Section , "Databases Supported by the Oracle JDBC Driver"
- Section , "Configuring Access to a Database Using the Oracle JDBC Driver"
- http://www.oracle.com/technology/tech/java/sqlj_jdbc/index.html

Type 4 JDBC Driver for SQL Server from DataDirect

Oracle Event Processing provides a Type 4 JDBC driver from DataDirect for high-performance JDBC access to the SQL Server database. The Type 4 JDBC driver is optimized for the Java environment, allowing you to incorporate Java technology and extend the functionality and performance of your existing system.

The Oracle Event Processing Type 4 JDBC drivers from DataDirect are proven drivers that:

- Support performance-oriented and enterprise functionality such as distributed transactions, savepoints, multiple open result sets and parameter metadata.
- Are Java EE Compatibility Test Suite (CTS) certified and tested with the largest JDBC test suite in the industry.
- Include tools for testing and debugging JDBC applications.

For more information, see:

- Section , "Databases Supported by the Type 4 JDBC Driver for SQL Server from DataDirect"
- Section , "Configuring Access to a Database Using the Type 4 JDBC Drivers from Data Direct"

Supported Databases

Oracle Event Processing server supports different databases depending on the type of JDBC driver you use:

- Section , "Databases Supported by the Oracle JDBC Driver"
- Section , "Databases Supported by the Type 4 JDBC Driver for SQL Server from DataDirect"

Databases Supported by the Oracle JDBC Driver

Using the Oracle JDBC driver, you can access the following Oracle databases:

- Oracle Database 11g release 2 (11.2)

For more information, see Section , "Oracle JDBC Driver".

Databases Supported by the Type 4 JDBC Driver for SQL Server from DataDirect

Using the SQL Server Type 4 JDBC Driver from DataDirect, you can access the following SQL Server databases:

- Microsoft SQL Server 2005
- Microsoft SQL Server 2000
- Microsoft SQL Server 2000 Desktop Engine (MSDE 2000)
- Microsoft SQL Server 2000 Enterprise Edition (64-bit)
- Microsoft SQL Server 7.0

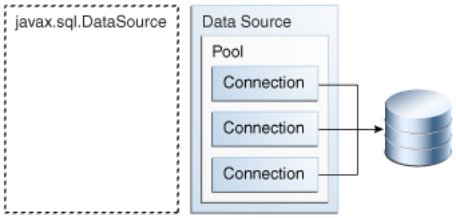
For more information, see Section , "Type 4 JDBC Driver for SQL Server from DataDirect".

Description of Oracle Event Processing Data Sources

Oracle Event Processing `DataSource` provides a JDBC data source connection pooling implementation that supports the Java Database Connectivity (JDBC 3.0) specification. Applications reserve and release `Connection` objects from a data source using the standard `DataSource.getConnection` and `Connection.close` APIs respectively.

Figure 13–1 shows the relationship between data source, connection pool, and `Connection` instances.

Figure 13–1 Oracle Event Processing Data Source



You must use the Oracle Event Processing server default data source or configure your own Oracle Event Processing `DataSource` in the server's `config.xml` file if you want to access a relational database:

- From an Oracle CQL processor `rule`.

  See "Configuring an Oracle CQL Processor Table Source" in the *Oracle Fusion Middleware Developer's Guide for Oracle Event Processing for Eclipse*.

- From an EPL processor `rule`.

  See "Configuring an EPL Processor" in the *Oracle Fusion Middleware Developer's Guide for Oracle Event Processing for Eclipse*.

- Event record and playback.

See “Storing Events in the Persistent Event Store” in the *Oracle Fusion Middleware Developer’s Guide for Oracle Event Processing for Eclipse*.

- From a cache loader or store.

  See “Exchanging Data Between a Cache and Another Data Source” in the *Oracle Fusion Middleware Developer’s Guide for Oracle Event Processing for Eclipse*.

You do not have to configure a `DataSource` in the server's `config.xml` file if you use the JDBC driver's API, such as `DriverManager`, directly in your application code.

For more information, see:

- Section , "Default Data Source Configuration"
- Section , "Custom Data Source Configuration"

Default Data Source Configuration

By default, the Oracle Event Processing server creates a local transaction manager. The transaction manager in turn depends on a configured RMI object, as described in Section , "rmi Configuration Object." Oracle Event Processing server guarantees that there will never be more than one transaction manager instance in the system.

If a database is unavailable at the time you start Oracle Event Processing server, by default, an Oracle Event Processing server data source retries every 10 seconds until it can create a connection. This allows the Oracle Event Processing server to successfully start up even if a database is unavailable. You can change the retry interval in the Oracle Event Processing server `config.xml` file using the `connection-pool-params` element `connection-creation-retry-frequency-seconds` child element. Setting this element to zero disables connection retry.

For more information, see Section , "Custom Data Source Configuration".

Custom Data Source Configuration

The Oracle Event Processing server `config.xml` file requires a configuration element for each data source that is to be created at runtime that references an external JDBC module descriptor.

This section describes how to configure a custom data source configuration. For more information on default data source configuration, see Section , "Default Data Source Configuration".

When you create an Oracle Event Processing domain using the Configuration Wizard, you can optionally configure a JDBC data source that uses one of the two DataDirect JDBC drivers; in this case the wizard updates the `config.xml` file for you. You configure the data source with basic information, such as the database you want to connect to and the connection username and password. You can also use the Configuration Wizard to update an existing server in a domain and add new data sources.

For more information, see:

- Section , "Creating an Oracle Event Processing Standalone-Server Domain"
- Section , "Creating an Oracle Event Processing Multi-Server Domain Using Oracle Coherence"
- Section , "Creating an Oracle Event Processing Multi-Server Domain Using Oracle Event Processing Native Clustering"

You can also update the `config.xml` file manually by adding a `data-source` element as Example 13–1 shows.

Example 13–1 Custom Data Source Configuration in Oracle Event Processing Server config.xml

```
<data-source>
    <name>rdbms</name>
    <data-source-params>
        <global-transactions-protocol>None</global-transactions-protocol>
    </data-source-params>
    <connection-pool-params>
        <test-table-name>SQL SELECT 1 FROM DUAL</test-table-name>
        <initial-capacity>5</initial-capacity>
        <max-capacity>10</max-capacity>
        <connection-creation-retry-frequency-seconds>
            60
        </connection-creation-retry-frequency-seconds>
    </connection-pool-params>
    <driver-params>
        <url>jdbc:oracle:thin:@localhost:5521:rdb</url>
        <driver-name>oracle.jdbc.OracleDriver</driver-name>
        <properties>
            <element><name>user</name><value>scott</value></element>
            <element><name>password</name><value>tiger</value></element>
        </properties>
        <use-xa-data-source-interface>true</use-xa-data-source-interface>
    </driver-params>
</data-source>
<transaction-manager>
    <name>TM</name>
    <rmi-service-name>RMI</rmi-service-name>
</transaction-manager>
```

A data source depends on the availability of a local transaction manager. You can rely on the default Oracle Event Processing server transaction manager or configure one using the `transaction-manager` element of `config.xml` as Example 13–1 shows. The transaction manager in turn depends on a configured RMI object, as described in Section , "rmi Configuration Object."

If a database is unavailable at the time you start Oracle Event Processing server, by default, an Oracle Event Processing server data source retries every 10 seconds until it can create a connection. This allows the Oracle Event Processing server to successfully start up even if a database is unavailable. Example 13–1 shows how you can change the retry interval in the Oracle Event Processing server `config.xml` file using the `connection-pool-params` element `connection-creation-retry-frequency-seconds` child element. Setting this element to zero disables connection retry.

For the full list of child elements of the `data-source` element, in particular the `connection-pool-params` and `data-source-params` elements, see "Server Configuration XSD Schema: wlevs_server_config.xsd" in the *Oracle Fusion Middleware Developer's Guide for Oracle Event Processing for Eclipse*.

For information on security configuration tasks that affect JDBC, see Section , "Configuring JDBC Security".

Getting the Native JDBC Connection

The *Oracle Fusion Middleware Oracle WebLogic Server API Reference* provides a WLConnection interface that contains useful methods for getting and manipulating Oracle data sources. For example, the following Java code gets the native Oracle database connection from the pooled connection object.

```
private DataSource ods;
private Connection wlConnection;
private OracleConnection connection;

wlConnection = ods.getConnection();
connection = (OracleConnection) ((WLConnection) wlConnection)
.getVendorConnection();
```

Note: Be sure to close pooled connections when you are finished with them, and do not use a native connection object after the pooled connection has been closed.

Configuring Access to a Database Using the Oracle JDBC Driver

The Oracle JDBC driver is automatically installed with Oracle Event Processing and ready to use. For more information, see Section , "Oracle JDBC Driver".

To configure access to a database using the Oracle JDBC driver:

1. Configure the data source in the server's `config.xml` file:
   a. To update the Oracle Event Processing server `config.xml` file using the Configuration Wizard, see Section , "Creating an Oracle Event Processing Standalone-Server Domain".
   b. To update the Oracle Event Processing server `config.xml` file manually, see Section , "Custom Data Source Configuration."

   Note: The `url` element for the Oracle JDBC driver is of the form:

   ```
   <url>jdbc:oracle:thin:@HOST:PORT:SID</url>
   ```

2. If Oracle Event Processing is running, restart it so it reads the new data source information.

   For more information, see Section , "Starting and Stopping Oracle Event Processing Servers".

Configuring Access to a Database Using the Type 4 JDBC Drivers from Data Direct

The type 4 JDBC drivers from DataDirect for SQL Server are automatically installed with Oracle Event Processing and ready to use. For more information, see Section , "Type 4 JDBC Driver for SQL Server from DataDirect".

To configure access to a database using the Type 4 JDBC drivers from Data Direct:

1. Configure the data source in the server's `config.xml` file:
   a. To update the Oracle Event Processing server `config.xml` file using the Configuration Wizard, see Section , "Creating an Oracle Event Processing Standalone-Server Domain".
   b. To update the Oracle Event Processing server `config.xml` file manually, see Section , "Custom Data Source Configuration."

Note: The `url` element for the type 4 JDBC drivers from DataDirect is of the form:

```
<url>jdbc:weblogic:sqlserver://HOST:PORT</url>
```

2. If Oracle Event Processing is running, restart it so it reads the new data source information.

   For more information, see Section , "Starting and Stopping Oracle Event Processing Servers".

Configuring Access to a Different Database Driver or Driver Version

In some cases, you may need to use a different version of the Oracle Database driver or Data Direct drivers than the version bundled with Oracle Event Processing or you may need to use a database driver other than the Oracle Database driver or Data Direct drivers.

This section describes the following:

- Section , "How to Access a Database Driver Using an Application Library Built With bundler.sh": this is the preferred method; you must create an OSGi bundle for your database driver.
- Section , "How to Access a Database Driver Using an Application Library Built With Oracle Event Processing IDE for Eclipse": this is the preferred method; you must create an OSGi bundle for your database driver.

  Use this option if you wish to manually configure the activator implementation.
- Section , "How to Access a Database Driver Using bootclasspath": this is an optional method; you do not need to create an OSGi bundle for your database driver.

How to Access a Database Driver Using an Application Library Built With bundler.sh

This procedure describes how to create an OSGi bundle for your driver using the bundler utility and deploy it on the Oracle Event Processing server.

This is the preferred method. If wish to manually configure the activator implementation, see Section , "How to Access a Database Driver Using an Application Library Built With Oracle Event Processing IDE for Eclipse".

For more information, see "Creating Application Libraries" in the *Oracle Fusion Middleware Developer's Guide for Oracle Event Processing for Eclipse*.

To access a database driver using an application library built with bundler.sh:

1. Set up your environment as described in "Setting Your Development Environment" in the *Oracle Fusion Middleware Developer's Guide for Oracle Event Processing for Eclipse*.
2. Execute the `bundler.sh` script to create an OSGi bundle containing your driver.

   The `bundler.sh` script is located in the *ORACLE_CEP_HOME*`/ocep_11.1/bin` directory, where *ORACLE_CEP_HOME* is the directory in which you installed the Oracle Event Processing server.

   Example 13–2 lists the `bundler.sh` command line options and Table 13–1 describes them.

Example 13–2 bundler.sh Command Line Options

```
bundler.sh
    -source <jar>
    -name <name>
    -version <version>
    [-factory <class>+]
    [-service <interface>+]
    [-stagedir <path>]
    [-targetdir <path>]
```

Table 13–1 bundler.sh Command Line Options

Argument	Description
`-source`	The path of the source JAR file to be bundled.
`-name`	The symbolic name of the bundle. The root of the target JAR file name is derived from the name value.
`-version`	The bundle version number. All exported packages are qualified with a version attribute with this value. The target JAR file name contains the version number.
`-factory`	An optional argument that specifies a space-delimited list of one or more factory classes that are to be instantiated and registered as OSGi services. Each service is registered with the OSGi service registry with name (`-name`) and version (`-version`) properties.
`-service`	An optional argument that specifies a space-delimited list of one or more Java interfaces that are used as the object class of each factory object service registration. If no interface names are specified, or the number of interfaces specified does not match the number of factory classes, then each factory object will be registered under the factory class name.
`-stagedir`	An optional argument that specifies where to write temporary files when creating the target JAR file. Default: `./bundler.tmp`
`-targetdir`	An optional argument that specifies the location of the generated bundle JAR file. Default: current working directory (`.`).

Example 13–3 shows how to use the `bundler.sh` to create an OSGi bundle for an Oracle JDBC driver.

Example 13–3 Using the Bundler Utility

```
bundler.sh \
    -source C:\drivers\com.oracle.ojdbc14_11.2.0.jar \
    -name oracle11g \
    -version 11.2.0 \
    -factory oracle.jdbc.xa.client.OracleXADataSource oracle.jdbc.OracleDriver \
    -service javax.sql.XADataSource java.sql.Driver \
    -targetdir C:\stage
```

The source JAR is an Oracle driver located in directory `C:\drivers`. The name of the generated bundle JAR is the concatenation of the `-name` and `-version` arguments (`oracle10g_11.2.0.jar`) and is created in the `C:\stage` directory. The bundle JAR contains the files that Example 13–4 shows.

Example 13–4 Bundle JAR Contents

```
   1465 Thu Jun 29 17:54:04 EDT 2006 META-INF/MANIFEST.MF
1540457 Thu May 11 00:37:46 EDT 2006 com.oracle.ojdbc14_11.2.0.jar
   1700 Thu Jun 29 17:54:04 EDT 2006 com/bea/core/tools/bundler/Activator.class
```

The command line options specify that there are two factory classes that will be instantiated and registered as an OSGi service when the bundle is activated, each under a separate object class as Table 13–2 shows.

Table 13–2 Factory Class and Service Interface

Factory Class	Service Interface
oracle.jdbc.xa.client.OracleXADataSource	javax.sql.XADataSource
oracle.jdbc.OracleDriver	java.sql.Driver

Each service registration will be made with a `name` property set to `oracle11g` and a `version` property with a value of `11.2.0`. Example 13–5 shows the Oracle Event Processing server log messages showing the registration of the services.

Example 13–5 Service Registration Log Messages

```
...
INFO: [Jun 29, 2006 5:54:18 PM] Service REGISTERED: { version=11.2.0, name=oracle11g,
objectClass=[ javax.sql.XADataSource ], service.id=23 }
INFO: [Jun 29, 2006 5:54:18 PM] Service REGISTERED: { version=11.2.0, name=oracle11g,
objectClass=[ java.sql.Driver ], service.id=24 }
INFO: [Jun 29, 2006 5:54:18 PM] Bundle oracle11g STARTED
...
```

3. Copy the bundler JAR to the Oracle Event Processing server library extensions directory.

   Because your Oracle Event Processing application is an application library which contains a driver, you copy it to the Oracle Event Processing server library extensions directory is the *DOMAIN_DIR*/*servername*/modules/ext directory, where *DOMAIN_DIR* refers to the domain directory such as /oracle_cep/user_projects/domains/mydomain and *servername* refers to the server instance, such as myserver. For example:

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

   For more information, see "Library Extensions Directory" in the *Oracle Fusion Middleware Developer's Guide for Oracle Event Processing for Eclipse*.

4. In the Oracle Event Processing server `config.xml` file, create a custom `data-source` element for your driver version and add a `driver-params` child element as Example 13–6 shows. For more information, see Section , "Oracle Event Processing Server Configuration Files".

Example 13–6 driver-params Child Element

```
<driver-params>
  <url>jdbc:oracle:thin:@lcw2k18:1531:lcw101</url>
  <driver-name>oracle.jdbc.xa.client.OracleXADataSource</driver-name
  <properties>
    <element>
      <name>user</name>
      <value>scott</value>
    </element>
    <element>
      <name>password</name>
      <value>{3DES}EoIfSBMhnW8=</value>
    </element>
    <element>
      <name>com.bea.core.datasource.serviceName</name>
      <value>oracle11g</value>
    </element>
    <element>
      <name>com.bea.core.datasource.serviceVersion</name>
```

```
      <value>11.2.0</value>
    </element>
    <element>
      <name>com.bea.core.datasource.serviceObjectClass</name>
      <value>javax.sql.XADataSource</value>
    </element>
  </properties>
  <use-xa-data-source-interface>true</use-xa-data-source-interface>
</driver-params>
```

Table 13–4 describes the relevant properties.

Table 13–3 driver-params Child Element Properties

Property	Description
`com.bea.core.datasource.serviceName`	Specifies the value of the `serviceName` registration property. This must match the `NAME` property in your `Activator` class.
`com.bea.core.datasource.serviceVersion`	Specifies the value of the `serviceVersion` registration property. This must match the `VERSION` property in your `Activator` class.
`com.bea.core.datasource.serviceObjectClass`	Specifies the interface name of the OSGI service registration.

For more information, see Section , "Custom Data Source Configuration".

5. Stop and start the Oracle Event Processing server.

   For more information, see Section , "Starting and Stopping Oracle Event Processing Servers".

How to Access a Database Driver Using an Application Library Built With Oracle Event Processing IDE for Eclipse

This procedure describes how to create an OSGi bundle for your driver using the Oracle Event Processing IDE for Eclipse and deploy it on the Oracle Event Processing server.

This is the preferred method. If do not wish to manually configure the activator implementation, see Section , "How to Access a Database Driver Using an Application Library Built With bundler.sh".

For more information, see “Creating Application Libraries” in the *Oracle Fusion Middleware Developer’s Guide for Oracle Event Processing for Eclipse*.

To access a database driver using an application library built with Oracle Event Processing IDE for Eclipse:

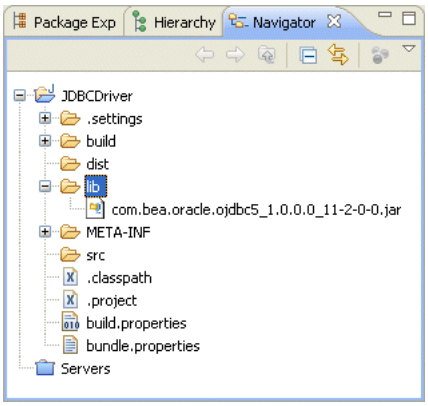
1. Using the Oracle Event Processing IDE for Eclipse, create a new Oracle Event Processing project.

   For more information, see “Creating Oracle Event Processing Projects” in the *Oracle Fusion Middleware Developer’s Guide for Oracle Event Processing for Eclipse*.

2. Right-click your project folder and select **New > Folder**.
3. Enter `lib` in the **Folder name** field and click **Finish**.
4. Outside of the Oracle Event Processing IDE for Eclipse, copy your JDBC JAR file into the `lib` folder.
5. Inside the Oracle Event Processing IDE for Eclipse, right-click the `lib` folder and select **Refresh**.

The JAR file appears in the `lib` folder as Figure 13–2 shows.

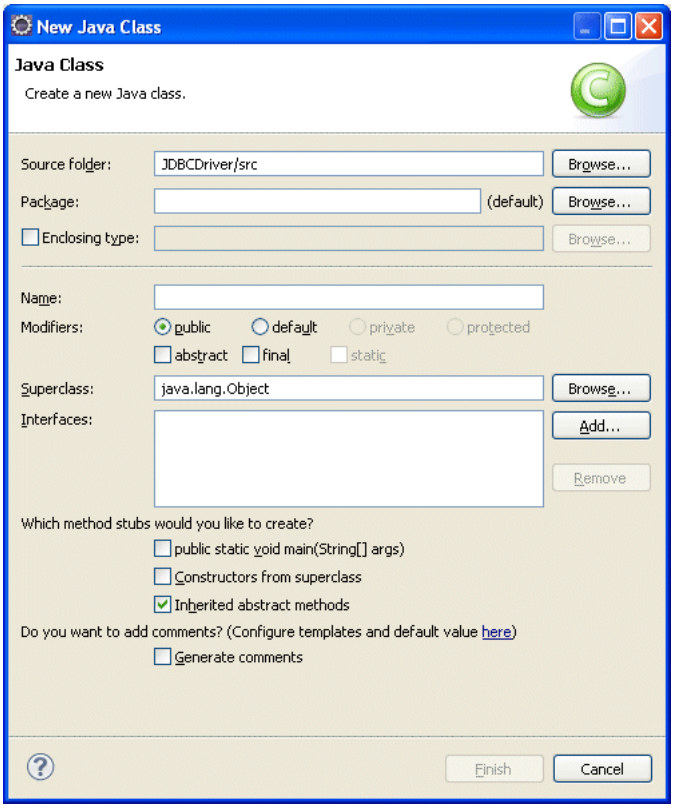
Figure 13–2 Oracle Event Processing IDE for Eclipse lib Directory



6. Right-click the **src** directory and select **New > Class**.

   The Java Class dialog appears as Figure 13–3 shows.

Figure 13–3 New Java Class Dialog



7. Configure the New Java Class dialog as Table 13–4 shows.

Table 13–4 New Java Class Parameters

Parameter	Description
Package	The package name. For example, `com.foo`.

Table 13–4 (Cont.) New Java Class Parameters

Parameter	Description
Name	The name of the class. For example, `MyActivator`.

Leave the other parameters at their default values.

8. Click **Finish**.

   A new Java class is added to your project.

9. Edit the Java class to implement it as Example 13–7 shows.

   Be sure to set the `NAME` and `VERSION` so that they supersede the existing version of JDBC driver. In this example, the existing version is:

   - `oracle10g`
   - `10.0.0`

   To supersede the existing version, the `MyActivator` class sets these values to:

   - `oracle11g`
   - `11.2.0`

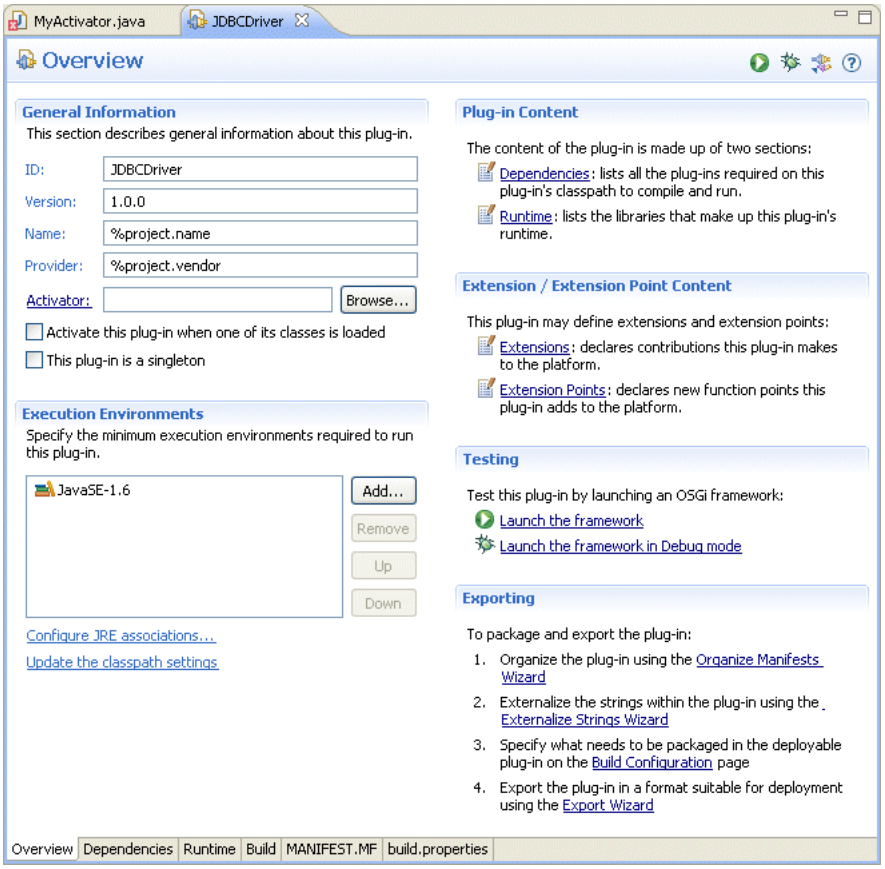
Example 13–7 MyActivator Class Implementation

```
package com.foo;

import java.util.Dictionary;
import java.util.Properties;

import javax.sql.XADataSource;
import java.sql.Driver;

import org.osgi.framework.BundleActivator;
import org.osgi.framework.BundleContext;
import org.osgi.framework.ServiceRegistration;

public class MyActivator implements BundleActivator {

  private static final String NAME="oracle11g";
  private static final String VERSION="11.2.0";

  private String[] factories =
{"oracle.jdbc.xa.client.OracleXADataSource","oracle.jdbc.OracleDriver"};
  private String[] interfaces= {"javax.sql.XADataSource","java.sql.Driver"};
  private ServiceRegistration[] serviceRegistrations = new
ServiceRegistration[factories.length];

  public void start(BundleContext bc) throws Exception {
    Dictionary props = new Properties();
    props.put("name", NAME);
    props.put("version", VERSION);
    for (int i=0; i<factories.length; i++) {
      Object svc = bc.getBundle().loadClass(factories[i]).newInstance();
      serviceRegistrations[i] = bc.registerService(interfaces[i], svc, props);
    }
  }

  public void stop(BundleContext bc) throws Exception {
    for (int i=0; i<serviceRegistrations.length; i++) {
      serviceRegistrations[i].unregister();
    }
  }
}
```

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

    The Manifest Editor appears as Figure 13–4 shows.

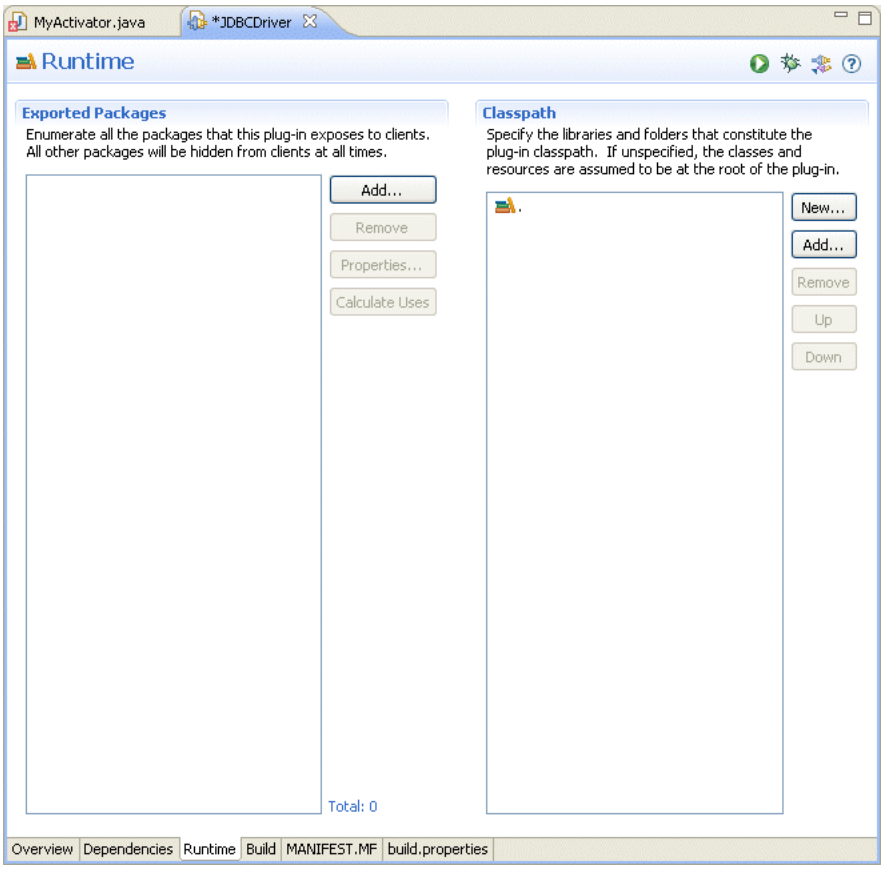
Figure 13–4 Manifest Editor: Overview Tab



11. Click the **Runtime** tab.

    The Runtime tab appears as Figure 13–5 shows.

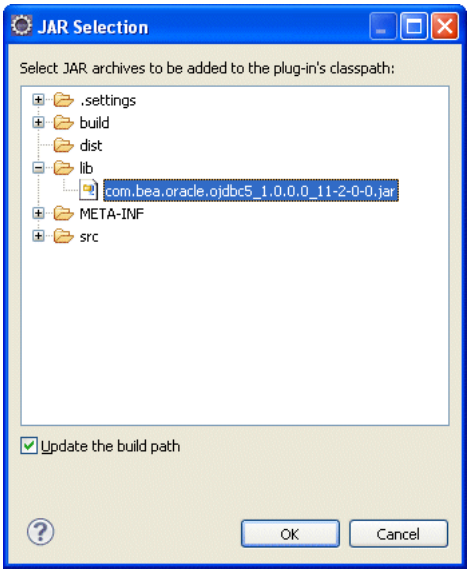
Figure 13–5 Manifest Editor: Runtime Tab



12. In the Classpath pane, click **Add**.

The JAR Selection dialog appears as Figure 13–6 shows.

Figure 13–6 JAR Selection Dialog

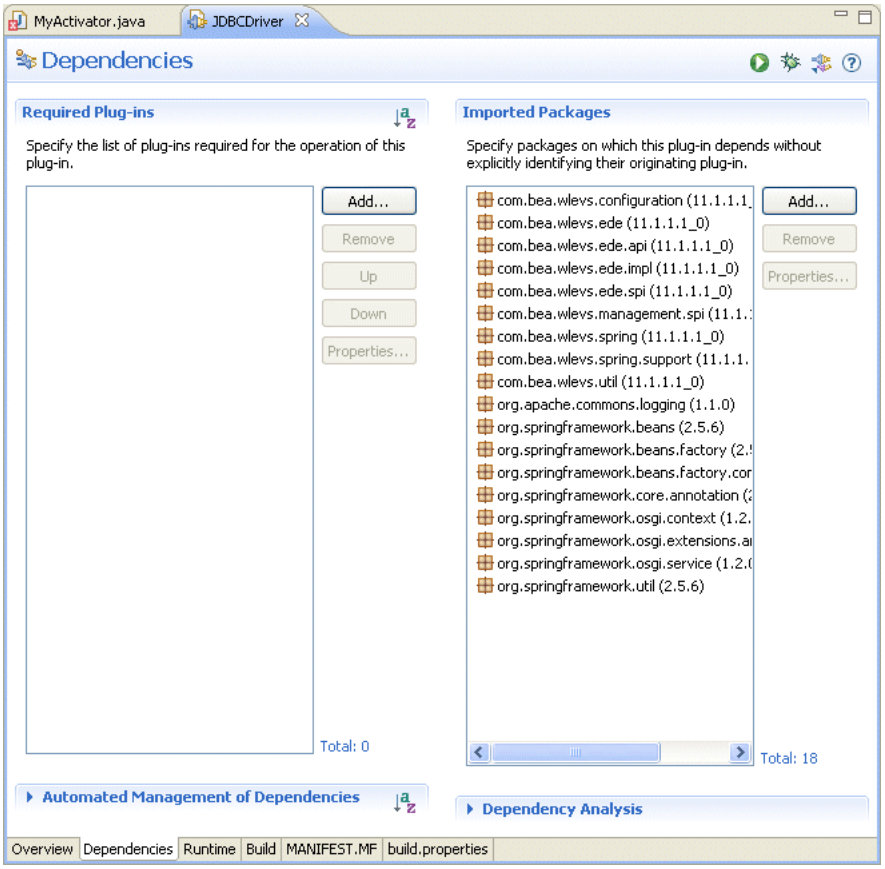


13. Expand the **lib** directory and select your database driver JAR file.

14. Click **OK**.
15. Click the **Dependencies** tab.

    The Dependencies tab appears as Figure 13–7 shows.

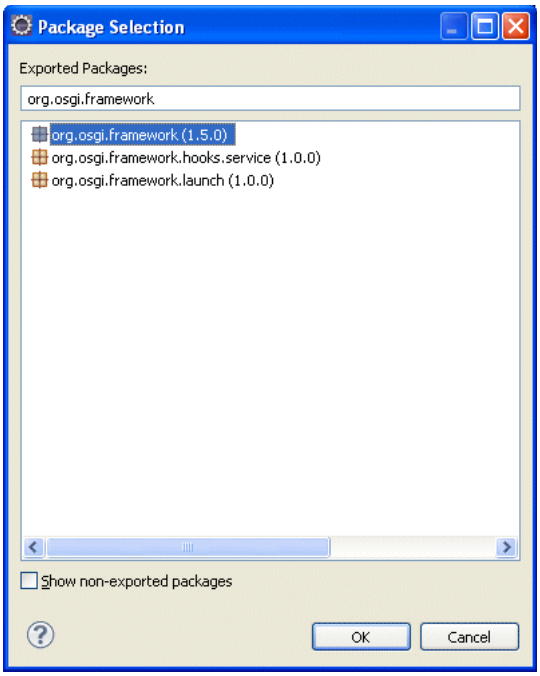
Figure 13–7 Manifest Editor: Dependencies Tab



16. In the Imported Packages pane, click **Add**.

    The Package Selection dialog appears as Figure 13–8 shows.

Figure 13–8 Package Selection Dialog



17. In the **Exported Packages** field, enter `org.osgi.framework`.

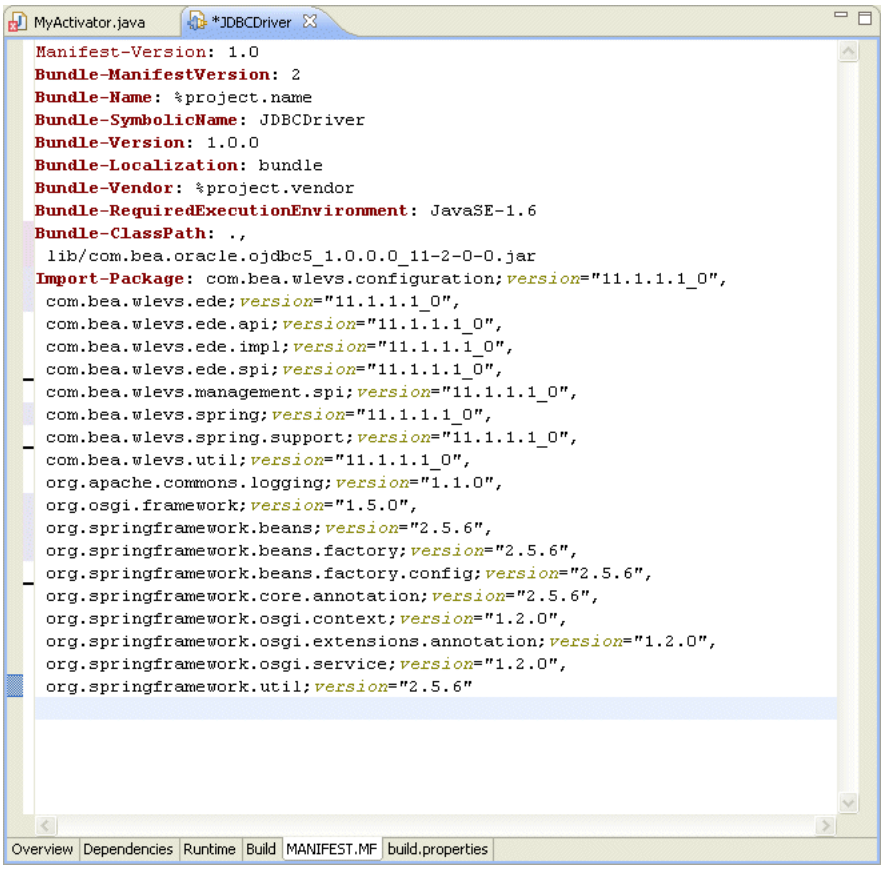
    The list box shows all the packages with that prefix as Figure 13–8 shows.

18. Select `org.osgi.framework` in the list box and click **OK**.

19. Click the **MANIFEST.MF** tab.

    The `MANIFEST.MF` tab appears as Figure 13–9 shows.

Figure 13–9 Manifest Editor



20. Un-JAR your database driver JAR to a temporary directory as Example 13–8 shows.

Example 13–8 Un-JAR the Database Driver

```
$ pwd
/tmp
$ ls com.*
com.bea.oracle.ojdbc6_1.0.0.0_11-1-0-7.jar
$ mkdir driver
$ cd driver
$ jar -xvf ../com.bea.oracle.ojdbc6_1.0.0.0_11-1-0-7.jar
$ ls
META-INF  oracle
$ cd META-INF
$ ls
MANIFEST.MF  services
```

21. Open your database driver JAR `MANIFEST.MF` file and copy its **Export-Package** entry and paste it into the Manifest Editor as Example 13–9 shows.

Example 13–9 Adding Export-Package to the Manifest Editor

```
Manifest-Version: 1.0
Bundle-ManifestVersion: 2
Bundle-Name: %project.name
Bundle-SymbolicName: JDBCDriver
Bundle-Version: 1.0.0
Bundle-Localization: bundle
```

```
Bundle-Vendor: %project.vendor
Bundle-RequiredExecutionEnvironment: JavaSE-1.6
Bundle-ClassPath: .
Import-Package: com.bea.wlevs.configuration;version="11.1.1.4_0", ...
Export-Package: oracle.core.lmx;version=1.0.0.0_11-1-0-7,oracle.core.l
 vf;version=1.0.0.0_11-1-0-7,oracle.jdbc;version=1.0.0.0_11-1-0-7,orac
 le.jdbc.aq;version=1.0.0.0_11-1-0-7,oracle.jdbc.connector;version=1.0
 .0.0_11-1-0-7,oracle.jdbc.dcn;version=1.0.0.0_11-1-0-7,oracle.jdbc.dr
 iver;version=1.0.0.0_11-1-0-7,oracle.jdbc.internal;version=1.0.0.0_11
 -1-0-7,oracle.jdbc.oci;version=1.0.0.0_11-1-0-7,oracle.jdbc.oracore;v
 ersion=1.0.0.0_11-1-0-7,oracle.jdbc.pool;version=1.0.0.0_11-1-0-7,ora
 cle.jdbc.rowset;version=1.0.0.0_11-1-0-7,oracle.jdbc.util;version=1.0
 .0.0_11-1-0-7,oracle.jdbc.xa;version=1.0.0.0_11-1-0-7,oracle.jdbc.xa.
 client;version=1.0.0.0_11-1-0-7,oracle.jpub.runtime;version=1.0.0.0_1
 1-1-0-7,oracle.net.ano;version=1.0.0.0_11-1-0-7,oracle.net.aso;versio
 n=1.0.0.0_11-1-0-7,oracle.net.jndi;version=1.0.0.0_11-1-0-7,oracle.ne
 t.ns;version=1.0.0.0_11-1-0-7,oracle.net.nt;version=1.0.0.0_11-1-0-7,
 oracle.net.resolver;version=1.0.0.0_11-1-0-7,oracle.security.o3logon;
 version=1.0.0.0_11-1-0-7,oracle.security.o5logon;version=1.0.0.0_11-1
 -0-7,oracle.sql;version=1.0.0.0_11-1-0-7,oracle.sql.converter;version
 =1.0.0.0_11-1-0-7
```

22. Add a `Bundle-Activator` element to the Manifest Editor as Example 13–10 shows.

    The value of the `Bundle-Activator` is the fully qualified class name of your `Activator` class.

Example 13–10 Adding a Bundle-Activator Element to the Manifest Editor

```
Manifest-Version: 1.0
Bundle-Activator: com.foo.MyActivator
Bundle-ManifestVersion: 2
Bundle-Name: %project.name
Bundle-SymbolicName: JDBCDriver
Bundle-Version: 1.0.0
Bundle-Localization: bundle
Bundle-Vendor: %project.vendor
Bundle-RequiredExecutionEnvironment: JavaSE-1.6
Bundle-ClassPath: .
Import-Package: com.bea.wlevs.configuration;version="11.1.1.4_0", ...
Export-Package: oracle.core.lmx;version=1.0.0.0_11-1-0-7, ...
...
```

23. Add a `DynamicImport-Package` element to the Manifest Editor as Example 13–11 shows.

Example 13–11 Adding a DynamicImport-Package Element to the Manifest Editor

```
Manifest-Version: 1.0
Bundle-Activator: com.foo.MyActivator
Bundle-ManifestVersion: 2
Bundle-Name: %project.name
Bundle-SymbolicName: JDBCDriver
Bundle-Version: 1.0.0
Bundle-Localization: bundle
Bundle-Vendor: %project.vendor
Bundle-RequiredExecutionEnvironment: JavaSE-1.6
Bundle-ClassPath: .
DynamicImport-Package: *
Import-Package: com.bea.wlevs.configuration;version="11.1.1.4_0", ...
Export-Package: oracle.core.lmx;version=1.0.0.0_11-1-0-7, ...
...
```

24. Export your Oracle Event Processing application to a JAR file.

For more information, see "How to Export an Oracle Event Processing Project" in the *Oracle Fusion Middleware Developer's Guide for Oracle Event Processing for Eclipse*.

Because your Oracle Event Processing application is an application library which contains a driver, you will copy your exported JAR to the Oracle Event Processing server library extensions directory.

The Oracle Event Processing server library extensions directory is the *DOMAIN_DIR*/*servername*/modules/ext directory, where *DOMAIN_DIR* refers to the domain directory such as /oracle_cep/user_projects/domains/mydomain and *servername* refers to the server instance, such as myserver. For example:

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

25. In the Oracle Event Processing server config.xml file, create a custom data-source element for your driver version and add a driver-params child element as Example 13–12 shows. For more information, see Section , "Oracle Event Processing Server Configuration Files".

Example 13–12 driver-params Child Element

```
<driver-params>
  <url>jdbc:oracle:thin:@lcw2k18:1531:lcw101</url>
  <driver-name>oracle.jdbc.xa.client.OracleXADataSource</driver-name
  <properties>
    <element>
      <name>user</name>
      <value>scott</value>
    </element>
    <element>
      <name>password</name>
      <value>{3DES}EoIfSBMhnW8=</value>
    </element>
    <element>
      <name>com.bea.core.datasource.serviceName</name>
      <value>oracle11g</value>
    </element>
    <element>
      <name>com.bea.core.datasource.serviceVersion</name>
      <value>11.2.0</value>
    </element>
    <element>
      <name>com.bea.core.datasource.serviceObjectClass</name>
      <value>javax.sql.XADataSource</value>
    </element>
  </properties>
  <use-xa-data-source-interface>true</use-xa-data-source-interface>
</driver-params>
```

Table 13–5 describes the relevant properties.

Table 13–5 driver-params Child Element Properties

Property	Description
com.bea.core.datasource.serviceName	Specifies the value of the serviceName registration property. This must match the NAME property in your Activator class.
com.bea.core.datasource.serviceVersion	Specifies the value of the serviceVersion registration property. This must match the VERSION property in your Activator class.
com.bea.core.datasource.serviceObjectClass	Specifies the interface name of the OSGI service registration.

For more information, see Section , "Custom Data Source Configuration".

26. Stop and start the Oracle Event Processing server.

    For more information, see Section , "Starting and Stopping Oracle Event Processing Servers".

How to Access a Database Driver Using bootclasspath

Optionally, you can use the bootclasspath to access your own JDBC driver with Oracle Event Processing.

Oracle recommends that you use an application library instead, as Section , "How to Access a Database Driver Using an Application Library Built With bundler.sh" or Section , "How to Access a Database Driver Using an Application Library Built With Oracle Event Processing IDE for Eclipse" describes.

To access a database driver using bootclasspath:

1. Update the server start script in the server directory of your domain directory so that Oracle Event Processing finds the appropriate JDBC driver JAR file when it boots up.

   The name of the server start script is `startwlevs.cmd` (Windows) or `startwlevs.sh` (UNIX), and the script is located in the server directory of your domain directory. The out-of-the-box sample domains are located in *ORACLE_CEP_HOME*`/ocep_11.1/samples/domains`, and the user domains are located in *ORACLE_CEP_HOME*`/user_projects/domains`, where *ORACLE_CEP_HOME* refers to the Oracle Event Processing installation directory, such as `d:\oracle_cep`.

   Update the start script by adding the `-Xbootclasspath/a` option to the Java command that executes the `wlevs_3.0.jar` file. Set the `-Xbootclasspath/a` option to the full pathname of the JDBC driver you are going to use.

   For example, if you want to use the Windows Oracle thin driver, update the `java` command in the start script as follows -- the updated section is shown in bold (split for readability; in practice, the command should be on one line):

   ```
   %JAVA_HOME%\bin\java -Dwlevs.home=%USER_INSTALL_DIR% -Dbea.home=%BEA_HOME%
   -Xbootclasspath/a:%USER_INSTALL_DIR%\bin\com.bea.oracle.ojdbc14_10.2.0.jar
   -jar "%USER_INSTALL_DIR%\bin\wlevs_3.0.jar" -disablesecurity %1 %2 %3 %4 %5 %6
   ```

   In the example, `%USER_INSTALL_DIR%` points to *ORACLE_CEP_HOME*`\ocep_11.1`.

2. Configure the data source in the server's `config.xml` file:

   a. To update the Oracle Event Processing server `config.xml` file using the Configuration Wizard, see Section , "Creating an Oracle Event Processing Standalone-Server Domain".

   b. To update the Oracle Event Processing server `config.xml` file manually, see Section , "Custom Data Source Configuration."

3. If Oracle Event Processing is running, restart it so it reads the new `java` option and data source information.

   For more information, see Section , "Starting and Stopping Oracle Event Processing Servers".

14

Configuring HTTP Publish-Subscribe for Oracle Event Processing

This chapter describes how to configure HTTP publish-subscribe (based on the Bayeux protocol) for use with Oracle Event Processing, including how to create and configure an HTTP publish-subscribe server and an example configuration.

This chapter includes the following sections:

Overview of HTTP Publish-Subscribe

An HTTP Publish-Subscribe Server (HTTP pub-sub server) is a mechanism whereby Web clients subscribe to channels and then publish messages to these channels using asynchronous messages over HTTP.

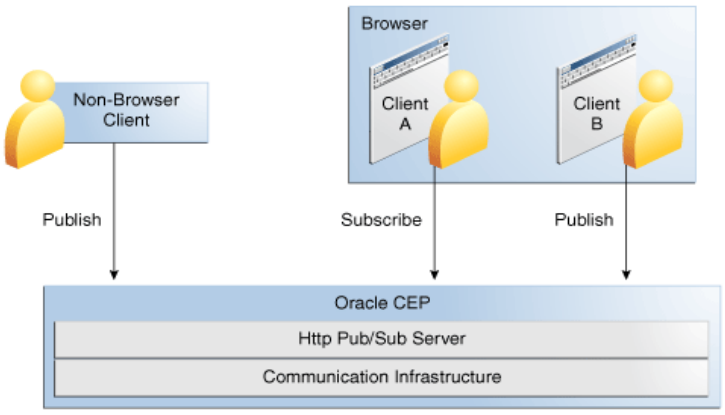
The simple request/response nature of a standard Web application requires that all communication be initiated by the client; this means that the server can only push updated data to its clients if it receives an explicit request. This mechanism is adequate for traditional applications in which data from the server is required only when a client requests it, but inadequate for dynamic real-time applications in which the server must send data even if a client has not explicitly requested it. The client can use the traditional HTTP pull approach to check and retrieve the latest data at regular intervals, but this approach is lacking in scalability and leads to high network traffic because of redundant checks. The HTTP Publish-Subscribe Server solves this problem by allowing clients to subscribe to a channel (similar to a topic in JMS) and receive messages as they become available.

The HTTP pub-sub server is based on the Bayeux protocol proposed by the cometd project. The Bayeux protocol defines a contract between the client and the server for communicating with asynchronous messages over HTTP. It allows clients to register and subscribe to channels, which are named destinations or sources of events. Registered clients, or the HTTP pub-sub server itself, then publishes messages to these channels which in turn any subscribed clients receive.

The HTTP pub-sub server can communicate with any client that can understand the Bayeux protocol. The HTTP pub-sub server is responsible for identifying clients, negotiating trust, exchanging Bayeux messages, and, most importantly, pushing event messages to subscribed clients.

Figure 14–1 describes the basic architecture of the HTTP pub-sub server included in Oracle Event Processing.

Figure 14–1 HTTP Publish-Subscribe Server in Oracle Event Processing



How the HTTP Pub-Sub Server Works

There is a one-to-one relationship between a servlet and an HTTP pub-sub server; in other words, each servlet has access to one unique HTTP pub-sub server. Each HTTP pub-sub server has its own list of channels. The servlet uses a context object to get a handle to its associated HTTP pub-sub server.

In Oracle Event Processing, HTTP pub-sub server instances are configured in the `config.xml` file of the server instance. System administrator uses the `config.xml` to configure the name of the HTTP pub-sub server, specify the transport and other parameters. You then use Oracle Event Processing Visualizer to add new channels and configure security for the channels.

Oracle Event Processing application developers can optionally use the built-in HTTP pub-sub adapters to publish and subscribe to channels within their applications. If, however, developers need the HTTP pub-sub server to perform additional steps, such as monitoring, collecting, or interpreting incoming messages from clients, then they must use the server-side HTTP pub-sub server APIs to program this functionality.

For Web 2.0 Ajax clients (such as Dojo) or rich internet applications (such as Adobe Flex) to communicate with the HTTP pub-sub server, the clients need a library that supports the Bayeux protocol. The Dojo JavaScript library provides four different transports, of which two are supported by the HTTP pub-sub server: long-polling and callback-polling.

HTTP Pub-Sub Server Support in Oracle Event Processing

Every Oracle Event Processing server includes a default HTTP pub-sub server. This server is used internally by Oracle Event Processing Visualizer and by the Record and Playback example. You can either use the default HTTP pub-sub server for your own Web 2.0 application or create a new one.

The default HTTP pub-sub server has the following properties:

- HTTP pub-sub server URL— `http://`*`host`*`:`*`port`*`/pubsub`, where *`host`* and *`port`* refer to the computer on which Oracle Event Processing is running and the port number to which it listens, respectively.
- Transport: Uses long-polling transport.

- Allows clients to publish messages to a channel without having explicitly connected to the HTTP pub-sub server.
- Includes the following three channels used internally by Oracle Event Processing Visualizer; do not delete these channels:
  - `/evsmonitor`
  - `/evsalert/`
  - `/evsdomainchange`

For details about configuring the default HTTP pub-sub server, or creating a new one, see:

- Section , "Creating a New HTTP Publish-Subscribe Server"
- Section , "Configuring an Existing HTTP Publish-Subscribe Server."
- Section , "Example HTTP Publish-Subscribe Server Configuration."

Oracle Event Processing also includes two built-in adapters that easily harness HTTP pub-sub server functionality in your applications. By adding these adapters to your application you can both publish messages or subscribe to a server to receive messages, using either the local HTTP pub-sub server or a remote one.

For more information, see “Using and Creating HTTP Publish-Subscribe Adapters” in the *Oracle Fusion Middleware Developer’s Guide for Oracle Event Processing for Eclipse*.

Creating a New HTTP Publish-Subscribe Server

The following procedure describes how to create a new HTTP pub-sub server. See Section , "Example HTTP Publish-Subscribe Server Configuration" for a full example from the `config.xml` of a configured HTTP pub-sub server.

To create a new HTTP publish-subscribe server:

1. If the Oracle Event Processing server is running, stop it.

   See Section , "Starting and Stopping Oracle Event Processing Servers".

2. Using your favorite XML editor, open the Oracle Event Processing server's `config.xml` file.

   This file is located in the *`DOMAIN_DIR`*`/`*`servername`*`/config` directory, where *`DOMAIN_DIR`* refers to the domain directory and *`servername`* refers to the name of the server, such as `/oracle_cep/user_projects/myDomain/defaultserver/config`.

3. Add an `http-pubsub` child element of the root `config` element of `config.xml`, with `name`, `path` and `pub-sub-bean` child elements, as shown in bold:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<ns2:config xmlns:ns2="http://www.bea.com/ns/wlevs/config/server">
    <domain>
        <name>myDomain</name>
    </domain>
    ....
    <http-pubsub>
      <name>myPubSubServer</name>
      <path>/myPath</path>
      <pub-sub-bean>
         ...
      </pub-sub-bean>
```

```
    </http-pubsub>
    ...
</ns2:config>
```

Set the `name` element to the internal name of the HTTP pub-sub server.

Set the `path` element to the string that you want to appear in the URL for connecting to the HTTP pub-sub server.

The next step describes the `pub-sub-bean` element.

4. Add `server-config` and `channels` child elements of the `pub-sub-bean` element:

```
<http-pubsub>
  <name>myPubSubServer</name>
  <path>/myPath</path>
  <pub-sub-bean>
    <server-config>
       ...
    </server-config>
    <channels>
       ...
    </channels>
  </pub-sub-bean>
</http-pubsub>
```

5. Update the `server-config` child element of the `pub-sub-bean` element with HTTP pub-sub server configuration as required.

   For the full list of possible elements, see "Server Configuration XSD Schema: wlevs_server_config.xsd" in the *Oracle Fusion Middleware Developer's Guide for Oracle Event Processing for Eclipse*.

   The following are the most common configuration options:

   - Add a `supported-transport` element to specify the transport.

     The format of this element is as follows:

```
<server-config>
    <supported-transport>
        <types>
            <element>long-polling</element>
        </types>
    </supported-transport>
...
</server-config>
```

     Oracle Event Processing server supports the following transports:

     – `long-polling`: Using this transport, the client requests information from Oracle Event Processing server and if Oracle Event Processing server does not have information available, it does not reply until it has. When the Oracle Event Processing server replies, the client typically sends another request immediately.

     – `callback-polling`: Use this transport for HTTP publish-subscribe applications using a cross domain configuration in which the browser downloads the page from one Web server (including the JavaScript code) and connects to another server as an HTTP publish-subscribe client. This is required by the Bayeux protocol. For more information on the Bayeux protocol, see `http://svn.cometd.org/trunk/bayeux/bayeux.html`.

- Add a `publish-without-connect-allowed` element to specify whether clients can publish messages without having explicitly connected to the HTTP pub-sub server; valid values are `true` or `false`:

```
<server-config>
...
    <publish-without-connect-allowed>true</publish-without-connect-allowed>
</server-config>
```

- Add a `work-manager` element to specify the name of the work manager that delivers messages to clients. The value of this element corresponds to the value of the `<name>` child element of the `work-manager` you want to assign.

```
<server-config>
...
    <work-manager>myWorkManager</work-manager>
</server-config>
```

  For more information, see "work-manager" in the *Oracle Fusion Middleware Developer's Guide for Oracle Event Processing for Eclipse*.

- Add a `client-timeout-secs` element to specify the number of seconds after which the HTTP pub-sub server disconnects a client if the client does has not sent back a connect/reconnect message.

```
<server-config>
...
    <client-timeout-secs>600</client-timeout-secs>
</server-config>
```

6. Update the `channels` child element with at least one channel pattern.

   Channel patterns always begin with a forward slash (`/`). Clients subscribe to these channels to either publish or receive messages. Add a channel pattern as shown:

```
<channels>
    <element>
        <channel-pattern>/mychannel</channel-pattern>
    </element>
</channels>
```

7. Save the `config.xml` file.
8. Start the Oracle Event Processing server.

   See Section , "Starting and Stopping Oracle Event Processing Servers".

9. Use Oracle Event Processing Visualizer to configure or add channels. See:
   - "Configuring HTTP Publish-Subscribe Server Channels" in the *Oracle Fusion Middleware Visualizer User's Guide for Oracle Event Processing*
10. Use Oracle Event Processing Visualizer to configure security for the channels. See:
    - "Configuring Security for the HTTP Publish-Subscribe Channels" in the *Oracle Fusion Middleware Visualizer User's Guide for Oracle Event Processing*
    - Section , "Configuring HTTP Publish-Subscribe Server Channel Security"

Configuring an Existing HTTP Publish-Subscribe Server

The following procedure describes how to configure an existing HTTP pub-sub server. See Section , "Example HTTP Publish-Subscribe Server Configuration" for a full example from the `config.xml` of a configured HTTP pub-sub server.

To configure an existing HTTP publish-subscribe server:

1. If the Oracle Event Processing server is running, stop it.

   See Section , "Starting and Stopping Oracle Event Processing Servers".

2. Using your favorite XML editor, open the Oracle Event Processing server's `config.xml` file.

   This file is located in the *`DOMAIN_DIR`*`/`*`servername`*`/config` directory, where *`DOMAIN_DIR`* refers to the domain directory and *`servername`* refers to the name of the server, such as `/oracle_cep/user_projects/myDomain/defaultserver/config`.

3. Search for the `http-pubsub` element that corresponds to the HTTP pub-sub server you want to configure. For example, the default HTTP pub-sub server is as follows:

   ```
   <http-pubsub>
       <name>pubsub</name>
       <path>/pubsub</path>
       <pub-sub-bean>
           <server-config>
           ...
   </http-pubsub>
   ```

4. Update the `server-config` child element of the `pub-sub-bean` element (which in turn is a child element of `http-pubsub`) with HTTP pub-sub server configuration as required.

   For the full list of possible elements, see "Server Configuration XSD Schema: wlevs_server_config.xsd" in the *Oracle Fusion Middleware Developer's Guide for Oracle Event Processing for Eclipse*.

   The following are the most common configuration options:

   - Add a `supported-transport` element to specify the transport.

     The format of this element is as follows:

     ```
     <server-config>
         <supported-transport>
             <types>
                 <element>long-polling</element>
             </types>
         </supported-transport>
     ...
     </server-config>
     ```

     Oracle Event Processing server supports the following transports:

     – `long-polling`: Using this transport, the client requests information from Oracle Event Processing server and if Oracle Event Processing server does not have information available, it does not reply until it has. When the Oracle Event Processing server replies, the client typically sends another request immediately.

  - `callback-polling`: Use this transport for HTTP publish-subscribe applications using a cross domain configuration in which the browser downloads the page from one Web server (including the JavaScript code) and connects to another server as an HTTP publish-subscribe client. This is required by the Bayeux protocol. For more information on the Bayeux protocol, see `http://svn.cometd.org/trunk/bayeux/bayeux.html`.

- Add a `publish-without-connect-allowed` element to specify whether clients can publish messages without having explicitly connected to the HTTP pub-sub server; valid values are `true` or `false`:

```
<server-config>
...
    <publish-without-connect-allowed>true</publish-without-connect-allowed>
</server-config>
```

- Add a `work-manager` element to specify the name of the work manager that delivers messages to clients. The value of this element corresponds to the value of the `name` child element of the `work-manager` you want to assign.

```
<server-config>
...
    <work-manager>myWorkManager</work-manager>
</server-config>
```

  For more information, see "work-manager" in the *Oracle Fusion Middleware Developer's Guide for Oracle Event Processing for Eclipse*.

- Add a `client-timeout-secs` element to specify the number of seconds after which the HTTP pub-sub server disconnects a client if the client does has not sent back a connect/reconnect message.

```
<server-config>
...
    <client-timeout-secs>600</client-timeout-secs>
</server-config>
```

5. Save the `config.xml` file.
6. Start the Oracle Event Processing server.

   See Section , "Starting and Stopping Oracle Event Processing Servers".
7. Use Oracle Event Processing Visualizer to configure or add channels. See:
   - "Configuring HTTP Publish-Subscribe Server Channels" in the *Oracle Fusion Middleware Visualizer User's Guide for Oracle Event Processing*
8. Use Oracle Event Processing Visualizer to configure security for the channels. See:
   - "Configuring Security for the HTTP Publish-Subscribe Channels" in the *Oracle Fusion Middleware Visualizer User's Guide for Oracle Event Processing*
   - Section , "Configuring HTTP Publish-Subscribe Server Channel Security"

Example HTTP Publish-Subscribe Server Configuration

The following snippet of the `config.xml` file shows the configuration of the default HTTP pub-sub server present in every Oracle Event Processing server:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<ns2:config xmlns:ns2="http://www.bea.com/ns/wlevs/config/server">
    <domain>
```

```
        <name>myDomain</name>
    </domain>
    ....
    <http-pubsub>
        <name>pubsub</name>
        <path>/pubsub</path>
        <pub-sub-bean>
            <server-config>
                <supported-transport>
                    <types>
                    <element>long-polling</element>
                    </types>
                </supported-transport>
                <publish-without-connect-allowed>true</publish-without-connect-allowed>
            </server-config>
            <channels>
                <element>
                    <channel-pattern>/evsmonitor</channel-pattern>
                </element>
                <element>
                    <channel-pattern>/evsalert</channel-pattern>
                </element>
                <element>
                    <channel-pattern>/evsdomainchange</channel-pattern>
                </element>
            </channels>
        </pub-sub-bean>
    </http-pubsub>
...
</ns2:config>
```

15

Configuring Logging and Debugging for Oracle Event Processing

This chapter describes how to configure logging and debugging for Oracle Event Processing, including using the Oracle Event Processing logging service, Log4j, the and Apache Commons logging API, as well as configuring debugging options.

This chapter includes the following sections:

Overview of Logging and Debugging Configuration

System administrators and developers configure logging output and filter log messages to troubleshoot errors or to receive notification for specific events.

The following tasks describe some logging configuration scenarios:

- Stop `DEBUG` and `INFO` messages from going to the log file.
- Allow `INFO` level messages from the HTTP subsystem to be published to the log file, but not to standard out.
- Specify that a handler publishes messages that are `WARNING` severity level or higher.
- Specify a default logging level for the entire Oracle Event Processing server, and then have a specific Oracle Event Processing module override the default logging level. For example, the default logging level of the server could be `WARNING` while the logging level of the CEP module is `DEBUG`.
- Configure a logging level for a user-application deployed to Oracle Event Processing. In this case, the application must use the Commons Apache Logging Framework if the application is required to output log messages into the single server-wide log file to which the modules of the server itself also log their messages.

Oracle Event Processing supports the following logging systems:

- Section , "Commons Apache Logging Framework"
- Section , "OSGi Framework Logger"

- Section , "Log4j Logger"

Oracle Event Processing also provides a variety of debugging options that you can enable and disable to help diagnose your Oracle Event Processing applications. See Section , "Configuring Oracle Event Processing Debugging Options".

Note: For information on Oracle Event Processing security auditor logging, see Section , "Configuring the Oracle Event Processing Security Auditor".

Note: For information how to parse message catalogs to validate and generate classes used for localizing text in log messages, see "Managing Log Message Catalogs" in the *Oracle Fusion Middleware Developer's Guide for Oracle Event Processing for Eclipse*.

Commons Apache Logging Framework

Oracle Event Processing provides a `commons-logging` interface. The interface provides `commons.logging.LogFactory` and `Log` interface implementations. It includes an extension of the `org.apache.commons.logging.LogFactory` class that acts as a factory to create an implementation of the `org.apache.commons.logging.Log` that delegates to the `LoggingService` in the logging module. The name of this default implementation is `weblogic.logging.commons.LogFactoryImpl`.

This section describes the following:

- Section , "Setting the Log Factory"
- Section , "Using Log Severity Levels"
- Section , "Log Files"
- Section , "Log Message Format"
- Section , "OSGi Framework Logger"

For more information, see `http://jakarta.apache.org/commons/logging/apidocs/index.html`.

Setting the Log Factory

The following provides information on setting the log factory using system properties:

- The highest priority is given to the system property `org.apache.commons.logging.LogFactory`.
- You can set logging from the command line using:

```
-Dorg.apache.commons.logging.LogFactory=weblogic.logging.commons.LogFactoryImpl
```

- You can programmatically implement the logging by:

```
import org.apache.commons.logging.LogFactory;
System.setProperty(
    LogFactory.FACTORY_PROPERTY,
    "weblogic.logging.commons.LogFactoryImpl"
);
```

- The `weblogic.logging.commons.LogFactoryImpl` is the default log factory, if not explicitly set.

- To use another logging implementation, you must use the standard commons logging factory implementation. The `org.apache.commons.logging.impl.LogFactoryImpl` implementation is available in the commons logging jar. For example:

  ```
  -Dorg.apache.commons.logging.LogFactory=org.apache.commons.logging.impl.LogFactoryImpl
  ```

  or the equivalent programming would be:

  ```
  System.setProperty(
      LogFactory.FACTORY_PROPERTY,
      "org.apache.commons.logging.impl.LogFactoryImpl"
  );
  ```

Using Log Severity Levels

Each log message has an associated severity level. The level gives a rough guide to the importance and urgency of a log message. Predefined severities, ranging from `TRACE` to `EMERGENCY`, are converted to a log level when dispatching a log request to the logger. A log level object can specify any of the following values, from lowest (left-most) to highest (right-most) impact:

`TRACE`, `DEBUG`, `INFO`, `NOTICE`, `WARNING`, `ERROR`, `CRITICAL`, `ALERT`, `EMERGENCY`

You can set a log severity level on the logger, the handler, and a user application. When set on the logger, none of the handlers receive an event which is rejected by the logger. For example, if you set the log level to `NOTICE` on the logger, none of the handlers will receive `INFO` level events. When you set a log level on the handler, the restriction only applies to that handler and not the others. For example, turning `DEBUG` off for the File Handler means no `DEBUG` messages will be written to the log file, however, `DEBUG` messages will be written to standard out.

Users (Oracle Event Processing module owners or owners of user applications) are free to define the names that represent the logging category type used by the Apache commons logging for individual modules. However if the category names are defined as package names then based on the naming convention, a logging level hierarchy is assumed by default. For example, if two modules name their logging category names `com.oracle.foo` and `com.oracle.foo.bar`, then `com.oracle.foo` becomes the root node of `com.oracle.foo.bar`. This way any logging level applied to parent node (`com.oracle.foo`) automatically applies to `com.oracle.foo.bar`, unless the child node overrides the parent.

In other words, if the logging severity is specified for a node, it is effective unless the severity is inherited from the nearest parent whose severity is explicitly configured. The root node is always explicitly configured, so if nothing else if set, then all the nodes inherit the severity from the root.

Table 15–1 lists the severity levels of log messages.

Table 15–1 Log Message Severity

Severity	Meaning
`TRACE`	Used for messages from the Diagnostic Action Library. Upon enabling diagnostic instrumentation of server and application classes, `TRACE` messages follow the request path of a method.
`DEBUG`	A debug message was generated.
`INFO`	Used for reporting normal operations, a low-level informational message.
`NOTICE`	An informational message with a higher level of importance.

Table 15–1 (Cont.) Log Message Severity

Severity	Meaning
WARNING	A suspicious operation or configuration has occurred but it might not affect normal operation.
ERROR	A user error has occurred. The system or application can handle the error with no interruption and limited degradation of service.
CRITICAL	A system or service error has occurred. The system can recover but there might be a momentary loss or permanent degradation of service.
ALERT	A particular service is in an unusable state while other parts of the system continue to function. Automatic recovery is not possible; the immediate attention of the administrator is needed to resolve the problem.
EMERGENCY	The server is in an unusable state. This severity indicates a severe system failure or panic.

The system generates many messages of lower severity and fewer messages of higher severity. For example, under normal circumstances, they generate many INFO messages and no EMERGENCY messages.

Log Files

By default, Oracle Event Processing server writes log messages to the server.log and consoleoutput.log files in the *ORACLE_CEP_HOME*/user_projects/domains/*DOMAIN_DIR*/*servername* directory, where *ORACLE_CEP_HOME* refers to the Oracle Event Processing installation directory (such as d:/oracle_cep), *DOMAIN_DIR* refers to the domain directory (such as my_domain), and *servername* refers to the server instance directory (such as server1).

For information on configuring log file attributes, see Section , "log-file".

For information on server.log file message format, see Section , "Format of Output to a Log File".

For information on consoleoutput.log file message format, see Section , "Format of Output to Console, Standard Out, and Standard Error".

Log Message Format

Oracle Event Processing server writes log messages in different formats depending on the type of log file it is writing to:

- Section , "Format of Output to a Log File"
- Section , "Format of Output to Console, Standard Out, and Standard Error"

For more information on log files, see Section , "Log Files".

Format of Output to a Log File The system writes a message to the specified log file consisting of a #### prefix, Timestamp, Severity, Subsystem, Server Name, Connection, Thread ID or User ID or Transaction ID, Message ID, and the Message, along with a stacktrace if any. Each attribute is contained between angle brackets.

The following is an example of a message in the server log file (split for readability; in practice, the message might be on one line):

```
####<Feb 25, 2009 10:23:32 AM EST> <Notice> <Deployment> <> <myServer>
<RMI TCP Connection(4)-141.144.123.236> <> <> <> <1235575412801> <BEA-2045000>
<The application bundle "Hello" was deployed successfully to file
[C:\OracleCEP\user_projects\domains\ocep_
domain\defaultserver\applications\Hello\Hello.jar]
with version 1235575412708>
```

Format of Output to Console, Standard Out, and Standard Error The system writes a message to the console, standard out, or standard error consisting of `Locale-formatted Timestamp`, `Severity`, `Subsystem`, `Message ID`, and `Message`.

The following is an example of how the message from the previous section would be printed to standard out (split for readability; in practice, the message might be on one line):

```
<Feb 25, 2009 10:23:32 AM EST> <Notice> <Deployment> <BEA-2045000>
<The application bundle "Hello" was deployed successfully to file
[C:\OracleCEP\user_projects\domains\ocep_
domain\defaultserver\applications\Hello\Hello.jar]
with version 1235575412708>
```

OSGi Framework Logger

Oracle Event Processing has a low-level framework logger that is started before the OSGi framework. It is used to report logging event deep inside the OSGi framework and function as a custom default for the logging subsystem before it is configured.

For example, a user may see some log message, which has lower level or severity than what is set in the `config.xml` but higher or equal to what is set on the Launcher command line on the console or in the log file. Until the logging subsystem has started, log messages come from the framework logger and use the framework logging level to filter messages.

Log4j Logger

Log4j is an open source tool developed for putting log statements in your application. Log4j has three main components:

- Section , "Loggers"
- Section , "Appenders"
- Section , "Layouts"

The Log4j Java logging facility was developed by the Jakarta Project of the Apache Foundation. See:

- The Log4j Project at `http://logging.apache.org/log4j/`.
- The Log4j API at `http://logging.apache.org/log4j/1.2/apidocs/index.html`.
- Short introduction to log4j at `http://logging.apache.org/log4j/1.2/manual.html`.

For more information, see Section , "Configuring Log4j Logging"

Loggers

Log4j defines a `Logger` class. An application can create multiple loggers, each with a unique name. In a typical usage of Log4j, an application creates a `Logger` instance for each application class that will emit log messages. Loggers exist in a namespace hierarchy and inherit behavior from their ancestors in the hierarchy.

Appenders

Log4j defines appenders (handlers) to represent destinations for logging output. Multiple appenders can be defined. For example, an application might define an appender that sends log messages to standard out, and another appender that writes log messages to a file. Individual loggers might be configured to write to zero or more

appenders. One example usage would be to send all logging messages (all levels) to a log file, but only `ERROR` level messages to standard out.

Layouts

Log4j defines layouts to control the format of log messages. Each layout specifies a particular message format. A specific layout is associated with each appender. This lets you specify a different log message format for standard out than for file output, for example.

Configuring the Oracle Event Processing Logging Service

You configure Oracle Event Processing logging service attributes using Oracle Event Processing Visualizer or by editing the Oracle Event Processing server `config.xml` file.

For more information on configuring logging using Oracle Event Processing Visualizer, see "Managing Logs" in the *Oracle Fusion Middleware Visualizer User's Guide for Oracle Event Processing*.

The `config.xml` file is located in the *`ORACLE_CEP_HOME`*`/user_projects/domains/`*`DOMAIN_DIR`*`/`*`servername`*`/config` directory, where *`ORACLE_CEP_HOME`* refers to the Oracle Event Processing installation directory (such as `d:/oracle_cep`), *`DOMAIN_DIR`* refers to the domain directory (such as `my_domain`), and *`servername`* refers to the server instance directory (such as `server1`).

Example 15–1 shows a typical Oracle Event Processing server `config.xml` file with logging elements.

Example 15–1 Oracle Event Processing Server config.xml File With Logging Configuration

```
<?xml version="1.0" encoding="UTF-8"?>
<!--Sample XML file generated by XMLSpy v2007 sp2 (http://www.altova.com)-->
<n1:config
    xsi:schemaLocation="http://www.bea.com/ns/wlevs/config/server wlevs_server_config.xsd"
    xmlns:n1="http://www.bea.com/ns/wlevs/config/server"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
...
        <name>myLogService</name>
        <log-file-config>myFileConfig</log-file-config>
        <stdout-config>myStdoutConfig</stdout-config>
        <logger-severity>Notice</logger-severity>
        <logger-severity-properties>
            <entry>
                <key>LifeCycle</key>
                <value>Notice</value>
            </entry>
            <entry>
                <key>Management</key>
                <value>Notice</value>
            </entry>
            <entry>
                <key>CQLProcessor</key>
                <value>Notice</value>
            </entry>
            <entry>
                <key>EplProcessor</key>
                <value>Notice</value>
            </entry>
            <entry>
                <key>Stream</key>
                <value>Notice</value>
            </entry>
```

```
            <entry>
                <key>Ede</key>
                <value>Notice</value>
            </entry>
            <entry>
                <key>Cache</key>
                <value>Notice</value>
            </entry>
            <entry>
                <key>Adapters</key>
                <value>Notice</value>
            </entry>
            <entry>
                <key>Spring</key>
                <value>Notice</value>
            </entry>
            <entry>
                <key>Channel</key>
                <value>Notice</value>
            </entry>
            <entry>
                <key>Recplay</key>
                <value>Notice</value>
            </entry>
            <entry>
                <key>Monitor</key>
                <value>Notice</value>
            </entry>
            <entry>
                <key>Server</key>
                <value>Notice</value>
            </entry>
            <entry>
                <key>EventTrace</key>
                <value>Notice</value>
            </entry>
            <entry>
                <key>Deployment</key>
                <value>Notice</value>
            </entry>
        </logger-severity-properties>
    </logging-service>
    <log-file>
        <name>myFileConfig</name>
        <rotation-type>none</rotation-type>
    </log-file>
    <log-stdout>
        <name>myStdoutConfig</name>
        <stdout-severity>Debug</stdout-severity>
    </log-stdout>

</n1:config>
```

The following sections provide information on configuring Oracle Event Processing logging:

- Section , "logging-service"
- Section , "log-file"
- Section , "log-stdout"
- Section , "Configuring Severity for an Individual Module"

logging-service

This section provides information on the `logging-service` element:

Table 15–2 Configuration Parameters for logging-service

Parameter	Type	Description
`name`	`String`	The name of this configuration object.
`log-file-config`	`String`	The configuration of the log file and its rotation policies. See Section , "log-file."
`stdout-config`	`String`	The name of the `stdout` configuration object used to configure `stdout` output. See Section , "log-stdout."
`logger-severity`	`String`	Defines the threshold importance of the messages that are propagated to the handlers. The default value is `Info`. To see `Debug` and `Trace` messages, configure the `logger-severity` to either `Debug` or `Trace`. Valid values are: `Emergency`, `Alert`, `Critical`, `Error`, `Warning`, `Notice`, `Info`, `Debug`, and `Trace`.
`logger-severity-properties`	One or more `<entry>` child elements.	List of name-value pairs, enclosed in an `<entry>` element, that list individual modules (package name, application name, class name, or component such as `CQLProcessor`) and their logging severity. These severities override the default severity of the Oracle Event Processing server. See Section , "Configuring Severity for an Individual Module."

log-file

This section provides information on the `log-file` element:

Table 15–3 Configuration Parameters for log-file

Parameter	Type	Description
`name`	`String`	The name of this configuration object.
`base-log-file-name`	`String`	The log file name. Default value is `server.log`.
`log-file-severity`	`String`	Specifies the least important severity of messages written to the log file. Default value is `Trace`. Valid values are: ▪ `Emergency` ▪ `Alert` ▪ `Critical` ▪ `Error` ▪ `Warning` ▪ `Notice` ▪ `Info` ▪ `Debug` ▪ `Trace`
`log-file-rotation-dir`	`String`	Specifies the directory where old rotated files are stored. If not set, the old files are stored in the same directory as the base log file.

Table 15–3 (Cont.) Configuration Parameters for log-file

Parameter	Type	Description
rotation-type	String	Specifies how rotation is performed based on size, time, or not at all. Valid values are: ■ bySize ■ byTime ■ none
rotation-time	String	The time in k:mm format, where k is the hour specified in 24 hour notation and mm is the minutes. Default is 00:00
rotation-time-span-factor	Long	Factor applied to the timespan to determine the number of milliseconds that becomes the frequency of time based log rotations. Default is 3600000.
rotated-file-count	Integer	Specifies the number of old rotated files to keep if number-of-files-limited is true. Default value is 7.
rotation-size	Integer	The size threshold, in KB, at which the log file is rotated. Default is 500.
rotation-time-span	Integer	Specifies the interval for every time-based log rotation. Default value is 24.
rotate-log-on-startup-enabled	Boolean	If true, the log file is rotated on startup. Default value is true.
number-of-files-limited	Boolean	If true, old rotated files are deleted. Default is false.

log-stdout

This section provides information on the log-stdout element:

Table 15–4 Configuration Parameters for log-stdout

Parameter	Type	Description
name	String	The name of this configuration object.
stdout-severity	String	The threshold severity for messages sent to stdout. Default value is Notice. Valid values are: ■ Emergency ■ Alert ■ Critical ■ Error ■ Warning ■ Notice ■ Info ■ Debug ■ Trace
stack-trace-depth	Integer	The number of stack trace frames to display on stdout. A default value of -1 means all frames are displayed.
stack-trace-enabled	Boolean	If true, stack traces are dumped to the console when included in logged messages. Default value is true.

Configuring Severity for an Individual Module

Individual modules of Oracle Event Processing can specify their logging severity. This severity overrides the default logging severity of Oracle Event Processing server.

You do this by specifying an `entry` child element in the `logger-severity-properties` element in the Oracle Event Processing server `config.xml` file. You may specify multiple `entry` child elements for any number of modules.

To configure severity for an individual module:

1. Edit the Oracle Event Processing server `config.xml` file.
2. Add an `entry` child element to the `logger-severity-properties` element as Example 15–2 shows.

Example 15–2 entry Child Element of the logger-severity-properties Element

```
<logging-service>
  <name>myLogService</name>
  <logger-severity>Warning</logger-severity>
  <logger-severity-properties>
  ...
      <entry>
          <key>CQLProcessor</key>
          <value>Debug</value>
      </entry>
  ...
  </logger-severity-properties>
  ...
</logging-service>
```

3. Set the `key` element to any of the following:
   - **Component name**: a component name constant exactly as Table 15–5 lists.

Table 15–5 Logging Component Name Constants

Component Name Constant	Description
`Adapters`	Applies to log messages from adapter instances running on the Oracle Event Processing server.
`Cache`	Applies to log messages from caching systems and cache instances running on the Oracle Event Processing server.
`Channel`	Applies to log messages from channels running on the Oracle Event Processing server.
`CQLProcessor`	Applies to log messages from Oracle CQL processors running on the Oracle Event Processing server.
`CQLServer`	Applies to log messages from the CQLEngine, which is at the core of each CQLProcessor.
`CQLServerTrace`	Applies to log messages from the CQLEngine, which is at the core of each CQLProcessor

Table 15–5 (Cont.) Logging Component Name Constants

Component Name Constant	Description
`Coherence`	Applies to log messages from Oracle Coherence, including messages related to clustering. The value you enter here is mapped to Oracle Coherence severity levels in the following way: Error: 1 Warning: 2 Notice: 3 Info: 4 Debug: 5 Trace: 9 You can customize logging from Oracle Coherence by overriding the `logging-config` setting in its configuration. For example, you can override the default log destination used for Oracle Coherence (log4j) and use another. For more information on overriding configuration, see Section , "Configuring the Oracle Coherence Cluster".
`EplProcessor`	Applies to log messages from EPL processors running on the Oracle Event Processing server.
`Ede`	Applies to log messages from the Event-Driven Environment, the Oracle Event Processing server event-dispatching infrastructure.

`EventTrace`	When set to `Info` or `Debug`, allows you to trace events as they flow through the EPN for all applications. You can dynamically change the severity of this log key using Oracle Event Processing Visualizer.
	At the `Info` severity, you see log messages like:
	`<May 26, 2009 5:53:49 PM PDT> <Info> <EventTrace> <BEA-000000> <Application [helloworld], Stage [helloworldOutputChannel] received insert event>`
	At the `Debug` severity, the log messages include details of the event:
	`<May 26, 2009 6:02:34 PM PDT> <Debug> <EventTrace> <BEA-000000> <Application [helloworld], Stage [helloworldOutputChannel] received insert event [HelloWorldEvent: HelloWorld - the current time is: 6:02:34 PM]>`

`Lifecycle`	Applies to log messages from Oracle Event Processing server and application lifecycle operations.
`Management`	Applies to log messages from Oracle Event Processing server general JMX-related management API operations.
`Monitor`	Applies to log messages from the Oracle Event Processing server monitoring service.
`Recplay`	Applies to log messages from Oracle Event Processing server event recording and playback operations.
`Spring`	Applies to log messages from Spring container operations.
`Stream`	Applies to log messages from stream instances running on the Oracle Event Processing server.

For example:

```
<entry>
```

```
    <key>CQLProcessor</key>
    <value>Debug</value>
</entry>
```

- **Application name**: the module name of any Oracle Event Processing server or user-defined application. For example:

```
<entry>
    <key>sample.HelloWorld</key>
    <value>Debug</value>
</entry>
```

- **Package name**: the name of any Oracle Event Processing server or user-supplied Java package. For example:

```
<entry>
    <key>com.bea.wlevs.ede</key>
    <value>Debug</value>
</entry>
```

  For more information on Oracle Event Processing server packages, see the *Oracle Fusion Middleware Java API Reference for Oracle Event Processing*.

- **Class name**: the fully qualified name of any Oracle Event Processing server or user-defined class. For example:

```
<entry>
    <key>com.bea.wlevs.cep.core.EPRuntimeImpl</key>
    <value>Debug</value>
</entry>
```

  For more information on Oracle Event Processing server classes, see the *Oracle Fusion Middleware Java API Reference for Oracle Event Processing*.

4. Set the `value` element to a severity level.

   See Section , "Using Log Severity Levels".

   For example:

```
<entry>
    <key>CQLProcessor</key>
    <value>Debug</value>
</entry>
```

   This severity level applies to the module you specified in the `key` element and overrides the default Oracle Event Processing server logging severity level set in the `logger-severity` element that Example 15–2 shows.

5. Repeat from step 2 for any other modules.
6. Save and close the `config.xml` file.

Configuring Log4j Logging

Oracle Event Processing supports the open-source log4j logging system.

This section describes the following tasks:

- Section , "Configuring log4j Properties"
- Section , "Configuring Application Manifest"
- Section , "Enabling Log4j Logging"

- Section , "Debugging Log4j Logging"

For more information, see Section , "Log4j Logger".

Configuring log4j Properties

The default configuration file is `log4j.properties`. It can be overridden by using the `log4j.configuration` system property. See https://www.qos.ch/shop/products/log4j/log4j-Manual.jsp.

The following is an example of a `log4j.properties` file:

Example 15–3 Example log4j.properties File

```
log4j.rootLogger=debug, R
log4j.appender.R=org.apache.log4j.RollingFileAppender
log4j.appender.R.File=D:/log4j/logs/mywebapp.log
log4j.appender.R.MaxFileSize=10MB
log4j.appender.R.MaxBackupIndex=10
log4j.appender.R.layout=org.apache.log4j.PatternLayout
log4j.appender.R.layout.ConversionPattern=%p %t %c - %m%n
log4j.logger=DEBUG, R
```

Configuring Application Manifest

Update the `MANIFEST.MF` file of your application to import the following required Log4j packages.

```
Import-Package:
    org.apache.log4j;version="1.2.13",
    org.apache.log4j.config;version="1.2.13",
    ...
```

Enabling Log4j Logging

To specify logging to a Log4j `Logger`, set the following system properties on the command line:

```
   -Dorg.apache.commons.logging.LogFactory=org.apache.commons.logging.impl.LogFact
oryImpl
   -Dorg.apache.commons.logging.Log=org.apache.commons.logging.impl.Log4JLogger
   -Dlog4j.configuration=<URL>/log4j.properties
```

Another very useful command line property is `-Dlog4j.debug=true`. Use this property when log4j output fails to appear or you get cryptic error messages.

Debugging Log4j Logging

If log4j output fails to appear or you get cryptic error messages, consider using the command line property `-Dlog4j.debug=true` on the command line.

For more information, see Section , "Enabling Log4j Logging".

Using the Apache Commons Logging API

You can use Apache Commons logging API in your Oracle Event Processing applications to log application-specific messages to the Oracle Event Processing `server.log` and `consoleoutput.log` files.

To use the commons logging API:

1. Set the system property `org.apache.commons.logging.LogFactory` to `weblogic.logging.commons.LogFactoryImpl`.

   This `LogFactory` creates instances of `weblogic.logging.commons.LogFactoryImpl` that implement the `org.apache.commons.logging.Log` interface.

2. From the `LogFactory`, get a reference to the Commons `Log` object by name.

   This name appears as the subsystem name in the log file.

3. Use the `Log` object to issue log requests to logging services.

   The Commons `Log` interface methods accept an object. In most cases, this will be a string containing the message text.

   The Commons `LogObject` takes a message ID, subsystem name, and a string message argument in its constructor. See org.apache.commons.logging at http://jakarta.apache.org/commons/logging/api/index.html.

4. The `weblogic.logging.commons.LogImpl` log methods direct the message to the server log.

Example 15–4 Commons Code Example

```
import org.apache.commons.logging.LogFactory;
import org.apache.commons.logging.Log;

public class MyCommonsTest {
  public void testCommonsLogging() {
    System.setProperty(LogFactory.FACTORY_PROPERTY,
      "weblogic.logging.commons.LogFactoryImpl");
    Log clog = LogFactory.getFactory().getInstance("MyCommonsLogger");
    // Log String objects
    clog.debug("Hey this is common debug");
    clog.fatal("Hey this is common fatal", new Exception());
    clog.error("Hey this is common error", new Exception());
    clog.trace("Dont leave your footprints on the sands of time");
  }
}
```

Configuring Oracle Event Processing Debugging Options

Table 15–6 lists the debugging options that Oracle Event Processing provides. You can enable and disable these debugging options to help diagnose problems with your Oracle Event Processing applications.

Table 15–6 Debug Flags

Debug Flag	Description
`com.bea.core.debug.DebugBootBundle`	Boot Debugging
`com.bea.core.debug.DebugBootBundle.stdout`	Boot Debugging debug strings go to stdout
`com.bea.core.debug.DebugCM`	Configuration Manager
`com.bea.core.debug.DebugCM.stdout`	Configuration Manager debug strings go to stdout
`com.bea.core.debug.DebugConfigurationRuntime`	Runtime information from the Runtime MBeans
`com.bea.core.debug.DebugCSS`	CSS
`com.bea.core.debug.DebugCSS.stdout`	CSS debug strings go to stdout
`com.bea.core.debug.DebugCSSServices`	CSS Services

Table 15–6 (Cont.) Debug Flags

Debug Flag	Description
com.bea.core.debug.DebugCSSServices.stdout	CSS Services debug strings go to stdout
com.bea.core.debug.DebugJDBCConn	JDBC Connection
com.bea.core.debug.DebugJDBCInternal	JDBC Internal
com.bea.core.debug.DebugJDBCRMI	JDBC RMI
com.bea.core.debug.DebugJDBCSQL	JDBC SQL
com.bea.core.debug.DebugJTA2PC	JTA 2PC
com.bea.core.debug.DebugJTA2PCDetail	JTA 2PCDetail
com.bea.core.debug.DebugJTA2PCStackTrace	JTA 2PCStackTrace
com.bea.core.debug.DebugJTAGateway	JTA Gateway
com.bea.core.debug.DebugJTAGatewayStackTrace	JTA GatewayStackTrace
com.bea.core.debug.DebugJTAHealth	JTA Health
com.bea.core.debug.DebugJTAJDBC	JTA JDBC
com.bea.core.debug.DebugJTALifecycle	JTA Lifecycle
com.bea.core.debug.DebugJTALLR	JTA LLR
com.bea.core.debug.DebugJTAMigration	JTA Migration
com.bea.core.debug.DebugJTANaming	JTA Naming
com.bea.core.debug.DebugJTANamingStackTrace	JTA NamingStackTrace
com.bea.core.debug.DebugJTANonXA	JTA NonXA
com.bea.core.debug.DebugJTAPropagate	JTA Propagate
com.bea.core.debug.DebugJTARecovery	JTA Recovery
com.bea.core.debug.DebugJTAResourceHealth	JTA ResourceHealth
com.bea.core.debug.DebugJTATLOG	JTA TLOG
com.bea.core.debug.DebugJTAXA	JTA XA
com.bea.core.debug.DebugJTAXAStackTrace	JTA XAStackTrace
com.bea.core.debug.DebugNetIO	NetIO
com.bea.core.debug.DebugOX	OSGi to JMX (OX)
com.bea.core.debug.DebugOX.stdout	OSGi to JMX (OX), debug goes to standard out.
com.bea.core.debug.DebugSCP	Simple Configuration Provider
com.bea.core.debug.DebugSCP.stdout	Simple Configuration Provider debug strings go to stdout
com.bea.core.debug.DebugSDS	Simple Declarative Services
com.bea.core.debug.DebugSDS.stdout	SDS debug strings go to stdout
com.bea.core.debug.DebugServiceHelper	Service Helper
com.bea.core.debug.DebugServiceHelper.stdout	Service Helper debug strings go to stdout
com.bea.core.debug.DebugStoreAdmin	Store Administration
com.bea.core.debug.DebugStoreIOLogical	Store IOLogical
com.bea.core.debug.DebugStoreIOLogicalBoot	Store IOLogicalBoot
com.bea.core.debug.DebugStoreIOPhysical	Store IOPhysical
com.bea.core.debug.DebugStoreIOPhysicalVerbose	Store IOPhysicalVerbose
com.bea.core.debug.DebugStoreXA	Store XA
com.bea.core.debug.DebugStoreXAVerbose	Store XAVerbose
com.bea.core.debug.servicehelper.dumpstack	Dump stack traces when Service Helper times out.

The following sections provide information on how to use these Oracle Event Processing debugging options:

- Section , "How to Configure Oracle Event Processing Debugging Options Using System Properties"
- Section , "How to Configure Oracle Event Processing Debugging Options Using a Configuration File"

If you are using Log4j logging, see also Section , "Debugging Log4j Logging".

How to Configure Oracle Event Processing Debugging Options Using System Properties

Use the following steps to configure debugging using system properties.

In this procedure, you will turn on Simple Declarative Services (SDS) debugging (`com.bea.core.debug.DebugSDS` from Table 15–6) using the Oracle Event Processing server `startwlevs.sh` file.

To configure Oracle Event Processing debugging options using system properties:

1. Locate the `DebugSDS` flag in Table 15–6:

   ```
   com.bea.core.debug.DebugSDS
   ```

2. Create a property by prepending `-D` to the flag:

   ```
   -Dcom.bea.core.debug.DebugSDS
   ```

3. Enable this debug flag by setting the property to `true`:

   ```
   -Dcom.bea.core.debug.DebugSDS=true
   ```

4. Start the Oracle Event Processing server using the `startwlevs.sh` with this property:

   ```
   ./startwlevs.sh -Dcom.bea.core.debug.DebugSDS=true
   ```

How to Configure Oracle Event Processing Debugging Options Using a Configuration File

Use the following steps to configure debugging from a configuration file.

In this procedure, you will turn on Simple Declarative Services (SDS) debugging (`com.bea.core.debug.DebugSDS` from Table 15–6) in the Oracle Event Processing server `config.xml` file.

To configure Oracle Event Processing debugging options using a configuration file:

1. Locate the `DebugSDS` flag in Table 15–6:

   ```
   com.bea.core.debug.DebugSDS
   ```

2. Create an XML tag by omitting the `com.bea.core.debug.` package name from the flag name:

   ```
   <DebugSDS></DebugSDS>
   ```

3. Edit the Oracle Event Processing server `config.xml` file and add a `debug` element with a `debug-properties` child element as Example 15–5 shows.
4. Add your `DebugSDS` element to the `debug-properties` element as Example 15–5 shows.
5. Enable this debug flag by setting the `DebugSDS` element to `true` as Example 15–5 shows.

Example 15–5 Creating a debug-properties Element for the Debug Flag

```
<config>
   <debug>
      <debug-properties>
         <DebugSDS>true</DebugSDS>
      </debug-properties>
   </debug>

</config>
```

6. Set `logger-severity` to `Debug` in the `logging-service` element as Example 15–6 shows.
7. Set `stdout-severity` to `Debug` in the `log-stdout` element as Example 15–6 shows.

Example 15–6 Enabling Debug Logging

```
<config>
   <debug>
      <debug-properties>
         <DebugSDS>true</DebugSDS>
      </debug-properties>
   </debug>

   <logging-service>
      <logger-severity>Debug</logger-severity>
      <stdout-config>logStdout</stdout-config>
      <log-file-config>logFile</log-file-config>
   </logging-service>

   <log-file>
   <name>logFile</name>
   <log-file-severity>Debug</log-file-severity>
   <number-of-files-limited>true</number-of-files-limited>
   <rotated-file-count>4</rotated-file-count>
   <rotate-log-on-startup-enabled>true</rotate-log-on-startup-enabled>
   </log-file>

   <log-stdout>
      <name>logStdout</name>
      <stdout-severity>Debug</stdout-severity>
   </log-stdout>
</config>
```

Part V

References

Part V contains the following appendices:

- Appendix A, "wlevs.Admin Command-Line Reference"
- Appendix B, "Deployer Command-Line Reference"
- Appendix C, "Security Utilities Command-Line Reference"

A

wlevs.Admin Command-Line Reference

This appendix provides a reference to the Oracle Event Processing `wlevs.Admin`utility, which you can use to administer Oracle Event Processing, dynamically configure rules for Oracle Continuous Query Language processors, and monitor event latency and throughput.

This appendix includes the following sections:

Overview of the wlevs.Admin Utility

The `wlevs.Admin` utility is a command-line interface to administer Oracle Event Processing and, in particular, dynamically configure the rules for Oracle CQL and EPL processors and monitor the event latency and throughput of an application. The utility internally uses JMX to query the configuration and runtime MBeans of both the Oracle Event Processing server and deployed applications.

The Oracle Event Processing configuration framework allows concurrent changes to both the application and Oracle Event Processing server configuration by multiple users. The framework does not use locking to manage this concurrency, but rather uses

optimistic version-based concurrency. This means that two users can always view the configuration of the same object with the intention to update it, but only one user is allowed to commit their changes. The other user will then get an error if they try to update the same configuration object, and must refresh their session to view the updated configuration.

Each `wlevs.Admin` utility command runs in its own transaction, which means that there is an implicit commit after each execution of a command. If you want to batch multiple configuration changes in a single transaction, you must use JMX directly to make these changes rather than the `wlevs.Admin` utility.

Configuring the wlevs.Admin Utility Environment

Before you can use the `wlevs.Admin` utility, you must configure your environment appropriately.

To configure the wlevs.Admin utility environment:

1. Install and configure the Oracle Event Processing software, as described in "Installing Oracle Event Processing" in the *Oracle Fusion Middleware Getting Started Guide for Oracle Event Processing*.
2. Configure JMX connectivity for the domain you want to administer. See Chapter 12, "Configuring JMX for Oracle Event Processing."
3. Open a command window and set your environment as described in "Setting Your Development Environment" in the *Oracle Fusion Middleware Developer's Guide for Oracle Event Processing for Eclipse*.
4. Set your `CLASSPATH` in one of the following ways:
   a. Implicitly set your `CLASSPATH` by using the `-jar` argument when you run the utility.

      Set the argument to the *`ORACLE_CEP_HOME`*`/ocep_11.1/bin/wlevsadmin.jar` file, where *`ORACLE_CEP_HOME`* refers to the main Oracle Event Processing installation directory.

      When you use the `-jar` argument, you do not specify the `wlevs.Admin` utility name at the command line. For example:

      ```
      prompt> java -jar d:/oracle_cep/ocep_11.1/bin/wlevsadmin.jar
      -url service:jmx:msarmi://localhost:9002/jndi/jmxconnector
      UPLOAD -application helloworld -processor helloworldProcessor
      -sourceURL file:///d:/test/newrules2.xml
      ```

   b. Explicitly update your `CLASSPATH` by adding the following files to the `CLASSPATH` environment variable:

      ```
      ORACLE_CEP_HOME/ocep_11.1/bin/wlevsadmin.jar
      ORACLE_CEP_HOME/ocep_11.1/bin/wlevs.jar
      ORACLE_CEP_HOME/ocep_11.1/modules/com.bea.wlevs.deployment.server_
      11.1.0.0.jar
      ORACLE_CEP_HOME/ocep_11.1/modules/com.bea.wlevs.ede_11.1.0.0.jar
      ORACLE_CEP_HOME/ocep_11.1/modules/com.bea.wlevs.management_11.1.0.0.jar
      ORACLE_CEP_HOME/modules/com.bea.core.jndi.context_6.0.0.0.jar
      ORACLE_CEP_HOME/modules/com.bea.core.jmx_6.0.0.0.jar
      ORACLE_CEP_HOME/modules/com.bea.core.rmi_6.0.0.0.jar
      ORACLE_CEP_HOME/modules/com.bea.core.i18n_1.4.0.0.jar
      ORACLE_CEP_HOME/modules/com.bea.core.diagnostics.core_2.1.0.0.jar
      ORACLE_CEP_HOME/modules/javax.xml.stream_1.1.1.0.jar
      ORACLE_CEP_HOME/com.bea.core.bootbundle_8.0.0.0.jar
      ```

Where *ORACLE_CEP_HOME* refers to the main directory into which you installed Oracle Event Processing.

Running the wlevs.Admin Utility Remotely

Sometimes it is useful to run the `wlevs.Admin` utility on a computer different from the computer on which Oracle Event Processing is installed and running.

To run the wlevs.Admin utility remotely:

1. Copy the following JAR files from the computer on which Oracle Event Processing is installed to the computer on which you want to run `wlevs.Admin`; you can copy the JAR files to the directory name of your choice:

   ORACLE_CEP_HOME`/ocep_11.1/bin/wlevsadmin.jar`
   ORACLE_CEP_HOME`/ocep_11.1/bin/wlevs.jar`
   ORACLE_CEP_HOME`/ocep_11.1/modules/com.bea.wlevs.deployment.server_11.1.0.0.jar`
   ORACLE_CEP_HOME`/ocep_11.1/modules/com.bea.wlevs.ede_11.1.0.0.jar`
   ORACLE_CEP_HOME`/ocep_11.1/modules/com.bea.wlevs.management_11.1.0.0.jar`
   ORACLE_CEP_HOME`/modules/com.bea.core.jndi.context_6.0.0.0.jar`
   ORACLE_CEP_HOME`/modules/com.bea.core.jmx_6.0.0.0.jar`
   ORACLE_CEP_HOME`/modules/com.bea.core.rmi_6.0.0.0.jar`
   ORACLE_CEP_HOME`/modules/com.bea.core.i18n_1.4.0.0.jar`
   ORACLE_CEP_HOME`/modules/com.bea.core.diagnostics.core_2.1.0.0.jar`
   ORACLE_CEP_HOME`/modules/javax.xml.stream_1.1.1.0.jar`

   Where *ORACLE_CEP_HOME* refers to the main directory into which you installed Oracle Event Processing.

2. Set your `CLASSPATH` in one of the following ways:

   - Implicitly set your `CLASSPATH` by using the `-jar` argument when you run the utility; set the argument to the *NEW_DIRECTORY*`/wlevsadmin.jar` file, where *NEW_DIRECTORY* refers to the directory on the remote computer into which you copied the required JAR files. When you use the `-jar` argument, you do not specify the `wlevs.Admin` utility name at the command line.
   - Explicitly update your `CLASSPATH` by adding all the files you copied to the remote computer to your `CLASSPATH` environment variable:

3. Invoke the `wlevs.Admin` utility as described in the next section.

Running wlevs.Admin Utility in SSL Mode

To use SSL when using the `wlevs.Admin` command-line utility, you must first create a trust keystore.

For more information, see Section , "SSL".

To run wlevs.Admin utility in SSL mode:

1. Open a command window and set your environment as described in “Setting Your Development Environment” in the *Oracle Fusion Middleware Developer’s Guide for Oracle Event Processing for Eclipse*.
2. If not already running, start the Oracle Event Processing server.

   See Section , "Starting and Stopping Oracle Event Processing Servers".

3. Change to the *DOMAIN_DIR*/*servername*/ssl directory, where *DOMAIN_DIR* refers to the main domain directory and *servername* refers to the name of your server.

   For example:

   d:\oracle_cep\user_projects\domains\mydomain\myserver\ssl.

4. Generate a trust keystore by specifying the following command (in practice, the command should be on one line):

```
prompt> java -classpath ORACLE_CEP_HOME\ocep_
11.1\common\lib\evspath.jar;ORACLE_CEP_HOME\ocep_
11.1\utils\security\wlevsgrabcert.jar
com.bea.wlevs.security.util.GrabCert host:secureport
-alias=alias truststorepath
```

   where

   - *ORACLE_CEP_HOME* refers to the Oracle Event Processing installation directory (such as d:/oracle_cep)
   - *host* refers to the computer on which server2 is running.
   - *secureport* refers to the SSL network i/o port configured for server2. Default value is 9003.

     For more information, see Example 10–5 in Section , "How to Configure SSL Manually."
   - *alias* refers to the alias for the certificate in the trust keystore. Default value is the hostname.
   - *truststorepath* refers to the full pathname of the generated trust keystore file; default is evstrust.jks

   For example (in practice, the command should be on one line):

```
prompt> java -classpath C:\OracleCEP\ocep_
11.1\common\lib\evspath.jar;C:\OracleCEP\ocep_
11.1\utils\security\wlevsgrabcert.jar
com.bea.wlevs.security.util.GrabCert server2:9003 -alias=server2 evstrust.jks
```

   To specify that the wlevs.Admin command-line utility use this trust keystore file, use the following properties:

   - -Djavax.net.ssl.trustStore—Name of the trust keystore file you created in the preceding step
   - -Djavax.net.ssl.trustStorePassword—Password of the trust keystore file.

   Also be sure to specify the secure port in the URL. For example:

```
prompt> java
-Djavax.net.ssl.trustStore=clitrust.jks
-Djavax.net.ssl.trustStorePassword=secret
-jar wlevsadmin.jar
-url service:jmx:msarmis://localhost:9003/jndi/jmxconnector
-username wlevs -password wlevs
SHUTDOWN -scheduleAt 600
```

Syntax for Invoking the wlevs.Admin Utility

The syntax for using the wlevs.Admin utility is as follows:

```
java wlevs.Admin
```

```
[ Connection Arguments ]
[ User Credentials Arguments ]
[ Common Arguments ]
COMMAND-NAME command-arguments
```

The command names and arguments are not case sensitive.

The following sections provide detailed syntax information about the arguments you can supply to the `wlevs.Admin` utility:

- Section , "Connection Arguments"
- Section , "User Credentials Arguments"
- Section , "Common Arguments"

The following sections provide detailed syntax information about the supported commands of the `wlevs.Admin` utility:

- Section , "Command for Getting Usage Help"
- Section , "Commands for Managing the Server Life Cycle"
- Section , "Commands for Managing the Oracle CQL Rules of an Application"
- Section , "Commands for Managing the EPL Rules of an Application"
- Section , "Commands for Managing Oracle Event Processing MBeans"
- Section , "Commands for Controlling Event Record and Playback"
- Section , "Commands for Monitoring Throughput and Latency"

Example Environment

In many of the examples throughout the sections that follow, it is assumed that a certain environment has been set up:

- The Oracle Event Processing instance listens to JMX requests on port `9002`.
- The Oracle Event Processing instance is installed on a host machine named `ariel` and uses this host name as its listen address.
- The `wlevs` username has system-administrator privileges and uses `wlevs` for a password.

Also, for clarity, all the examples are shown on multiple lines; however, when you run the command, enter all arguments and commands on a single line.

Exit Codes Returned by wlevs.Admin

All `wlevs.Admin` commands return an exit code of `0` if the command succeeds and an exit code of `1` if the command fails.

To view the exit code from a Windows command prompt, enter `echo %ERRORLEVEL%` after you run a `wlevs.Admin` command. To view the exit code in a `bash` shell, enter `echo $?`.

`wlevs.Admin` calls `System.exit(1)` if an exception is raised while processing a command, causing Ant and other Java client JVMs to exit.

Connection Arguments

```
java wlevs.Admin
    [ {-url URL} | {-listenAddress hostname -listenPort port} ]
```

```
[ User Credentials Arguments ]
[ Common Arguments ]
COMMAND-NAME command-arguments
```

When you invoke most `wlevs.Admin` commands, you specify the arguments in Table A–1 to connect to an Oracle Event Processing instance.

Table A–1 Connection Arguments

Argument	Definition
`-url service:jmx:msarmi://host:port/jndi/jmxconnector`	Specifies the URL that establishes a JMX connection to the Oracle Event Processing instance you want to administer, where: ▪ *host* refers to the name of the computer on which the Oracle Event Processing instance is running ▪ *port* refers to the Oracle Event Processing server JNDI port If you use this argument, do not specify `-listenAddress` or `-listenPort`. Other than *host* you specify the remainder of the URL as written. For example, if Oracle Event Processing is running on a computer with hostname ariel, and the JMX listening port is 9002, then the URL would be: `-url service:jmx:msarmi://ariel:9002/jndi/jmxconnector` See Chapter 12, "Configuring JMX for Oracle Event Processing" for details about configuring JMX, JNDI, and RMI for Oracle Event Processing.
`-listenAddress hostname`	Specifies the name of computer on which the Oracle Event Processing instances is running. This argument, together with `-listenPort`, is used to build the URL that establishes a JMX connection to the server you want to administer. You use this argument, together with `-listenPort`, *instead* of `-url`. For example, if Oracle Event Processing is running on a computer with hostname `ariel`, then this argument would be: `-listenAddress ariel`

`-listenPort port`	Specifies the port configured for Oracle Event Processing that listens to JMX connections. This argument, together with `-listenAddress`, is used to build the URL that establishes a JMX connection to the server you want to administer.
	You use this argument, together with `-listenAddress`, *instead* of `-url`.
	The JMX port is configured in the `config.xml` file of the Oracle Event Processing domain you are administering. In particular, the port is the `<port>` child element of the `<netio>` element, as shown:
	`<netio>`
	`<name>NetIO</name>`
	`<port>9002</port>`
	`</netio>`
	In the example, the port is `9002` and you specify as an argument as follows:
	`-listenPort 9002`
	See Chapter 12, "Configuring JMX for Oracle Event Processing" for details about configuring JMX, JNDI, and RMI for Oracle Event Processing.

User Credentials Arguments

```
java wlevs.Admin
    [ Connection Arguments ]
    [ -username username [-password password] ]
    [ Common Arguments ]
    COMMAND-NAME command-arguments
```

When you invoke most `wlevs.Admin` commands, you specify the arguments in Table A–2 to provide the user credentials of an Oracle Event Processing user who has permission to invoke the command.

If security has not been enabled for your Oracle Event Processing domain, then you do not have to provide user credentials.

Table A–2 User Credentials Arguments

Argument	Definition
-username *username*	The name of the user who is issuing the command. This user must have appropriate permission to view or modify the target of the command.
-password *password*	The password that is associated with the username.

Note: The exit code for all commands is 1 if the `wlevs.Admin` utility cannot connect to the server or if the Oracle Event Processing instance rejects the username and password.

Common Arguments

```
java wlevs.Admin
    [ Connection Arguments ]
    [ User Credentials Arguments ]
    [ -verbose ]
    COMMAND-NAME command-arguments
```

All `wlevs.Admin` commands support the argument in Table A–3 to get verbose output.

Table A–3 Common Arguments

Argument	Definition
-verbose	Specifies that `wlevs.Admin` should output additional verbose information.

Command for Getting Usage Help

This section describes the command for getting usage help.

HELP

Provides syntax and usage information for all Oracle Event Processing commands (by default) or for a single command if a command value is specified on the HELP command line.

You can issue this command from any computer on which the Oracle Event Processing is installed. You do not need to start a server instance to invoke this command, nor do you need to supply user credentials, even if security is enabled for the server.

Syntax

```
java wlevs.Admin HELP [COMMAND]
```

The *COMMAND* argument can be:

- The keyword `ALL`, which returns usage information about all commands.
- One of the keywords `MBEAN`, `RULES`, or `LIFECYCLE`, which returns usage information about the three different groups of commands.

- An actual command, such as `UPLOAD`, which returns usage information about the particular command.

Example

In the following example, information about using the UPLOAD command is requested:

```
prompt> java wlevs.Admin HELP UPLOAD
```

The command returns the following (in practice, Java commands should be on one line):

```
Description:
Uploads rules to be configured in the processor.

Usage:
java wlevs.Admin
        [-url | -listenAddress <host-name> -listenPort <port>]
        -username <username> -password <password>
        UPLOAD -application <application name> -processor <processor name> -sourceURL "source
url"

Where:
-application = Name of the application.
-processor = Name of the processor.
-sourceURL = source URL containing the rules in an XML format.


java wlevs.Admin -url service:jmx:msarmi://localhost:9002/jndi/jmxconnector
-username wlevs -password wlevs UPLOAD -application myapplication
-processor processor -sourceURL file:/d:/test/rules.xml
```

Commands for Managing the Server Life Cycle

Table A–4 is an overview of commands that manage the life cycle of a server instance. Subsequent sections describe command syntax and arguments, and provide an example for each command.

Table A–4 Overview of Commands for Managing the Server Life Cycle

Command	Description
SHUTDOWN	Gracefully shuts down a WebLogic Event Server.

SHUTDOWN

Gracefully shuts down the specified Oracle Event Processing instance.

A graceful shutdown gives Oracle Event Processing time to complete certain application processing currently in progress.

The `-url` connection argument specifies the particular Oracle Event Processing instance that you want to shut down, based on the `host` and `jmxport` values. See Section , "Connection Arguments" for details.

Syntax

```
java wlevs.Admin
    [ Connection Arguments ]
    [ User Credentials Arguments ]
    [ Common Arguments ]
    SHUTDOWN [-scheduleAt seconds]
```

Table A–5 SHUTDOWN Arguments

Argument	Definition
`-scheduleAt` *seconds*	Specifies the number of seconds after which the Oracle Event Processing instance shuts down. If you do not specify this parameter, the server instance shuts down immediately.

Example

The following example instructs the specified Oracle Event Processing instance to shut down in ten minutes:

```
prompt> java wlevs.Admin
       -url service:jmx:msarmi://localhost:9002/jndi/jmxconnector
       -username wlevs -password wlevs
        SHUTDOWN -scheduleAt 600
```

After you issue the command, the server instance prints messages to its log file and to its standard out. The messages indicate that the server state is changing and that the shutdown sequence is starting.

Commands for Managing the Oracle CQL Rules of an Application

Table A–6 is an overview of commands that manage the Oracle CQL rules for a particular processor of an Oracle Event Processing application. Subsequent sections describe command syntax and arguments, and provide an example for each command.

Table A–6 Overview of Commands for Managing Application Oracle CQL Rules

Command	Description
GETRULE	Returns the text of an existing Oracle CQL rule, query, or view of the processor of an Oracle Event Processing application.
ADDRULE	Adds a new Oracle CQL rule, query, or view to the processor of an Oracle Event Processing application.
DELETERULE	Deletes an existing Oracle CQL rule, query, or view from the processor of an Oracle Event Processing application.
REPLACERULE	Replaces an existing Oracle CQL rule, query, or view with new Oracle CQL text.
STARTRULE	Starts a previously stopped Oracle CQL rule or query.
STOPRULE	Stops a previously started Oracle CQL rule or query.
UPLOAD	Configures a set of Oracle CQL rules, queries, or views for a processor of an Oracle Event Processing application by uploading the rules from a component configuration XML file.
DOWNLOAD	Downloads the set of Oracle CQL rules, queries, or views associated with a processor of an Oracle Event Processing application to a component configuration XML file.

GETRULE

Returns the full text of an Oracle CQL rule, query, or view from the specified Oracle CQL processor of an Oracle Event Processing application.

Syntax

```
java wlevs.Admin
    [ Connection Arguments ]
    [ User Credentials Arguments ]
```

```
[ Common Arguments ]
GETRULE -application application -processor processor -rule rulename
```

Table A–7 GETRULE Arguments

Argument	Definition
`-application application`	Specifies the name of the Oracle Event Processing application whose Oracle CQL rules you want to manage. To get the exact name of your application, you can: ▪ Use `wlevs.Admin` to query for the name (see Section , "Querying for Application and Processor Names"). ▪ Use the Oracle Event Processing Visualizer: Start the Oracle Event Processing Visualizer (see "Starting the Oracle Event Processing Visualizer" in the *Oracle Fusion Middleware Visualizer User's Guide for Oracle Event Processing*). In the left pane, navigate to and expand the **Applications** node of the Oracle Event Processing instance to which the application is deployed. Each node under the Applications node is named with the exact application name. ▪ Look at the `MANIFEST.MF` file of the application; the application name is specified by the `Bundle-SymbolicName` header.
`-processor processor`	Specifies the name of the particular Oracle CQL processor, attached to the Oracle Event Processing application specified with the `-application` argument, whose Oracle CQL rules you want to manage. See Section , "Querying for Application and Processor Names" for details on getting the exact name if you do not know it.
`-rule rulename`	Specifies the name of the Oracle CQL rule, query, or view you want to see. See Section , "Querying for Application and Processor Names" for details on querying for the rule, query, or view name if you do not know it. You can also use the DOWNLOAD command to get the list of rules for a particular processor.

Example

The following example shows how to get the full text of the Oracle CQL view called `myview` from the Oracle CQL `helloworldProcessor` of the `helloworld` application:

```
prompt> java wlevs.Admin
        -url service:jmx:msarmi://localhost:9002/jndi/jmxconnector
        -username wlevs -password wlevs
        GETRULE -application helloworld -processor helloworldProcessor
        -rule myview
```

ADDRULE

Adds a new Oracle CQL rule, query, or view to the specified processor of an Oracle Event Processing application.

If a rule, query, or view with the same name (identified with the *`rulename`*, *`queryname`*, or *`viewname`* parameter) already exists, then the `ADDRULE` command replaces the existing rule, query, or view with the new one.

Note: An Oracle CQL query will immediately begin outputting events if its input channels provide input events. If you plan to use a query selector on a channel with an upstream Oracle CQL processor, you may observe unwanted query results on the downstream channel between the time you add the query to the upstream Oracle CQL processor and the time you configure the query selector on the downstream channel. For more information, see "Channel Properties: Outbound Channel With Query Selector" in the *Oracle Fusion Middleware Visualizer User's Guide for Oracle Event Processing*.

Syntax

```
java wlevs.Admin
    [ Connection Arguments ]
    [ User Credentials Arguments ]
    [ Common Arguments ]
    ADDRULE -application application -processor processor
        -rule  [rulename]  rulestring  |
        -query [queryname] querystring |
        -view  [viewname]  viewstring [-schema comma-separated-names]
        [-active true | false]
```

Table A–8 ADDRULE Arguments

Argument	Definition
`-application` *application*	Specifies the name of the Oracle Event Processing application whose Oracle CQL rules you want to manage. To get the exact name of your application, you can: ■ Use `wlevs.Admin` to query for the name (see Section , "Querying for Application and Processor Names"). ■ Use the Oracle Event Processing Visualizer: Start the Oracle Event Processing Visualizer (see "Starting the Oracle Event Processing Visualizer" in the *Oracle Fusion Middleware Visualizer User's Guide for Oracle Event Processing*). In the left pane, navigate to and expand the **Applications** node of the Oracle Event Processing instance to which the application is deployed. Each node under the Applications node is named with the exact application name. ■ Look at the `MANIFEST.MF` file of the application; the application name is specified by the `Bundle-SymbolicName` header.
`-processor` *processor*	Specifies the name of the particular processor, attached to the Oracle Event Processing application specified with the `-application` argument, whose Oracle CQL rules you want to manage. See Section , "Querying for Application and Processor Names" for details on getting the exact name if you do not know it.
`-rule [`*rulename*`]` *rulestring*	Specifies the Oracle CQL rule you want to add to the specified processor of your application. An Oracle CQL rules include: ■ `REGISTER|CREATE FUNCTION` (aggregate and single-row functions) ■ `REGISTER|CREATE WINDOW` The *rulename* parameter is not required; if you do not specify it, Oracle Event Processing generates a name for you. Enter the Oracle CQL *rulestring* using double quotes.
`-query [`*queryname*`]` *querystring*	Specifies the Oracle CQL query you want to add to the specified processor of your application. The *queryname* parameter is not required; if you do not specify it, Oracle Event Processing generates a name for you. Enter the Oracle CQL *querystring* using double quotes.

Table A–8 (Cont.) ADDRULE Arguments

Argument	Definition
`-view [viewname] viewstring [-schema comma-separated-names]`	Specifies the Oracle CQL view you want to add to the specified processor of your application. The *viewname* parameter is not required; if you do not specify it, Oracle Event Processing generates a name for you. Enter the Oracle CQL *viewstring* using double quotes. The *comma-separated-names* parameter is not required; if you do not specify it, Oracle Event Processing generates the schema based on the select statement in the *viewstring*.
`-active true | false`	Specifies if the rule should be started and ready to process events after being added. Valid values for this argument are `true` (start rule after adding) or `false` (do not start rule after adding); default value is `true`. If set to `false`, use STARTRULE to start the rule.

Example

The following example shows how to add the Oracle CQL query `SELECT * FROM Withdrawal [Rows 5]`, with name `myquery`, to the Oracle CQL processor `helloworldProcessor` of the `helloworld` application:

```
prompt> java wlevs.Admin
        -url service:jmx:msarmi://localhost:9002/jndi/jmxconnector
        -username wlevs -password wlevs
        ADDRULE -application helloworld -processor helloworldProcessor
        -query myquery "SELECT * FROM Withdrawal [Rows 5]"
```

DELETERULE

Deletes an existing Oracle CQL rule from the specified processor of an Oracle Event Processing application.

Syntax

```
java wlevs.Admin
    [ Connection Arguments ]
    [ User Credentials Arguments ]
    [ Common Arguments ]
    DELETERULE -application application -processor processor -rule rulename
```

Table A–9 DELETERULE Arguments

Argument	Definition
`-application application`	Specifies the name of the Oracle Event Processing application whose Oracle CQL rules you want to manage. To get the exact name of your application, you can: ■ Use `wlevs.Admin` to query for the name (see Section , "Querying for Application and Processor Names"). ■ Use the Oracle Event Processing Visualizer: Start the Oracle Event Processing Visualizer (see "Starting the Oracle Event Processing Visualizer" in the *Oracle Fusion Middleware Visualizer User's Guide for Oracle Event Processing*). In the left pane, navigate to and expand the **Applications** node of the Oracle Event Processing instance to which the application is deployed. Each node under the Applications node is named with the exact application name. ■ Look at the `MANIFEST.MF` file of the application; the application name is specified by the `Bundle-SymbolicName` header.

Table A–9 (Cont.) DELETERULE Arguments

Argument	Definition
`-processor processor`	Specifies the name of the particular processor, attached to the Oracle Event Processing application specified with the `-application` argument, whose Oracle CQL rules, queries, and views you want to manage. See Section , "Querying for Application and Processor Names" for details on getting the exact name if you do not know it.
`-rule rulename`	Specifies the name of the Oracle CQL rule, query, or view you want to delete. See Section , "Querying for Application and Processor Names" for details on querying for the rule name if you do not know it. You can also use the DOWNLOAD command to get the list of rules, queries, or views for a particular Oracle CQL processor.

Example

The following example shows how to delete the Oracle CQL view called `myview` from the Oracle CQL `helloworldProcessor` of the `helloworld` application:

```
prompt> java wlevs.Admin
        -url service:jmx:msarmi://localhost:9002/jndi/jmxconnector
        -username wlevs -password wlevs
        DELETERULE -application helloworld -processor helloworldProcessor -rule myview
```

REPLACERULE

Replaces an existing Oracle CQL rule, query, or view with another rule, query, or view. Oracle Event Processing first destroys the original rule, query, or view and then inserts the new one in its place.

Syntax

```
java wlevs.Admin
    [ Connection Arguments ]
    [ User Credentials Arguments ]
    [ Common Arguments ]
    REPLACERULE -application application -processor processor
        -rule rulename rulestring
```

Table A–10 REPLACERULE Arguments

Argument	Definition
`-application application`	Specifies the name of the Oracle Event Processing application whose Oracle CQL rules you want to manage. To get the exact name of your application, you can: ▪ Use `wlevs.Admin` to query for the name (see Section , "Querying for Application and Processor Names"). ▪ Use the Oracle Event Processing Visualizer: Start the Oracle Event Processing Visualizer (see "Starting the Oracle Event Processing Visualizer" in the *Oracle Fusion Middleware Visualizer User's Guide for Oracle Event Processing*). In the left pane, navigate to and expand the **Applications** node of the Oracle Event Processing instance to which the application is deployed. Each node under the Applications node is named with the exact application name. ▪ Look at the `MANIFEST.MF` file of the application; the application name is specified by the `Bundle-SymbolicName` header.

Table A–10 (Cont.) REPLACERULE Arguments

Argument	Definition
`-processor` *processor*	Specifies the name of the particular Oracle CQL processor, attached to the Oracle Event Processing application specified with the `-application` argument, whose Oracle CQL rules you want to manage. See Section , "Querying for Application and Processor Names" for details on getting the exact name if you do not know it.
`-rule` *rulename rulestring*	Specifies the name of the Oracle CQL rule, query, or view you want to replace. Oracle Event Processing deletes the old rule, query, or view and then inserts a new one, with the same name but with the new rule text. In the case of a view, Oracle Event Processing generates the schema based on the select statement in the *rulestring*. Enter the Oracle CQL *rulestring* using double quotes.

Example

The following example shows how to replace an Oracle CQL query called `myquery` with the Oracle CQL text `SELECT * FROM Withdrawal [Rows 10]` in the Oracle CQL `helloworldProcessor` of the `helloworld` application:

```
prompt> java wlevs.Admin
        -url service:jmx:msarmi://localhost:9002/jndi/jmxconnector
        -username wlevs -password wlevs
        REPLACERULE -application helloworld -processor helloworldProcessor
        -rule myquery "SELECT * FROM Withdrawal [Rows 10]"
```

STARTRULE

Starts an existing Oracle CQL rule or query that was previously stopped in the specified processor of an Oracle Event Processing application.

Syntax

```
java wlevs.Admin
    [ Connection Arguments ]
    [ User Credentials Arguments ]
    [ Common Arguments ]
    STARTRULE -application application -processor processor -rule rulename
```

Table A–11 STARTRULE Arguments

Argument	Definition
`-application` *application*	Specifies the name of the Oracle Event Processing application whose Oracle CQL rules you want to manage. To get the exact name of your application, you can: ▪ Use `wlevs.Admin` to query for the name (see Section , "Querying for Application and Processor Names"). ▪ Use the Oracle Event Processing Visualizer: Start the Oracle Event Processing Visualizer (see "Starting the Oracle Event Processing Visualizer" in the *Oracle Fusion Middleware Visualizer User's Guide for Oracle Event Processing*). In the left pane, navigate to and expand the **Applications** node of the Oracle Event Processing instance to which the application is deployed. Each node under the Applications node is named with the exact application name. ▪ Look at the `MANIFEST.MF` file of the application; the application name is specified by the `Bundle-SymbolicName` header.

Table A–11 (Cont.) STARTRULE Arguments

Argument	Definition
`-processor processor`	Specifies the name of the particular processor, attached to the Oracle Event Processing application specified with the `-application` argument, whose Oracle CQL rules, queries, and views you want to manage. See Section , "Querying for Application and Processor Names" for details on getting the exact name if you do not know it.
`-rule rulename`	Specifies the name of the Oracle CQL rule or query you want to start. **NOTE**: You cannot stop and start a view. Views are always active. See Section , "Querying for Application and Processor Names" for details on querying for the rule name if you do not know it. You can also use the DOWNLOAD command to get the list of rules, queries, or views for a particular Oracle CQL processor.

Example

The following example shows how to start the Oracle CQL query called `myquery` from the Oracle CQL `helloworldProcessor` of the `helloworld` application:

```
prompt> java wlevs.Admin
        -url service:jmx:msarmi://localhost:9002/jndi/jmxconnector
        -username wlevs -password wlevs
        STARTRULE -application helloworld -processor helloworldProcessor -rule myquery
```

STOPRULE

Stops an existing Oracle CQL rule or query that was previously started in the specified processor of an Oracle Event Processing application.

Syntax

```
java wlevs.Admin
    [ Connection Arguments ]
    [ User Credentials Arguments ]
    [ Common Arguments ]
    STOPRULE -application application -processor processor -rule rulename
```

Table A–12 STOPRULE Arguments

Argument	Definition
`-application application`	Specifies the name of the Oracle Event Processing application whose Oracle CQL rules or queries you want to manage. See Section , "Querying for Application and Processor Names" for details on using `wlevs.Admin` to get the exact name of your application if you do not currently know it. You can also get the exact application name by looking at the `MANIFEST.MF` file of the application; the application name is specified by the `Bundle-SymbolicName` header.
`-processor processor`	Specifies the name of the particular processor, attached to the Oracle Event Processing application specified with the `-application` argument, whose Oracle CQL rules, queries, and views you want to manage. See Section , "Querying for Application and Processor Names" for details on getting the exact name if you do not know it.
`-rule rulename`	Specifies the name of the Oracle CQL rule or query you want to stop. **NOTE**: You cannot stop and start a view. Views are always active. See Section , "Querying for Application and Processor Names" for details on querying for the rule name if you do not know it. You can also use the DOWNLOAD command to get the list of rules, queries, or views for a particular Oracle CQL processor.

Example

The following example shows how to stop the Oracle CQL query called `myquery` from the Oracle CQL `helloworldProcessor` of the `helloworld` application:

```
prompt> java wlevs.Admin
        -url service:jmx:msarmi://localhost:9002/jndi/jmxconnector
        -username wlevs -password wlevs
        STOPRULE -application helloworld -processor helloworldProcessor -rule myquery
```

UPLOAD

Replaces the configured Oracle CQL rules for a specified processor with the Oracle CQL rules from an uploaded component configuration file.

The component configuration file that contains the list of Oracle CQL rules conforms to the component configuration file schema (see "Component Configuration XSD Schema: wlevs_application_config.xsd" in the *Oracle Fusion Middleware Developer's Guide for Oracle Event Processing for Eclipse*). This file contains one or more Oracle CQL rules that will replace those currently configured for the specified processor. An example of such a component configuration file is as follows:

```
<?xml version="1.0" encoding="UTF-8" ?>
<config>
  <processor>
  <name>helloworldProcessor</name>
    <rules>
      <query id="helloworldRule1">
          <![CDATA[  SELECT * FROM HelloWorldEvent [Rows 2] ]]>
      </query>
    </rules>
  </processor>
</config>
```

In the preceding example, the component configuration file configures a single Oracle CQL query, with name `helloworldRule1`, and its Oracle CQL query text is `SELECT * FROM HelloWorldEvent [Rows 2]`.

Caution: When you use the `UPLOAD` command of the `wlevs.Admin` utility, you use the `-processor` argument to specify the name of the Oracle CQL processor to which you want to add the Oracle CQL rules, as you do with the other Oracle CQL commands. This means that the utility *ignores* any `<name>` elements in the component configuration file to avoid any naming conflicts.

For details and examples of creating the component configuration file, see "Configuring Oracle CQL Processors" in the *Oracle Fusion Middleware Developer's Guide for Oracle Event Processing for Eclipse*.

You an obtain a copy of a processor's component configuration file using the `DOWNLOAD` command as Section , "DOWNLOAD" describes.

Syntax

```
java wlevs.Admin
    [ Connection Arguments ]
    [ User Credentials Arguments ]
    [ Common Arguments ]
    UPLOAD -application application -processor processor -sourceURL sourcefileURL
```

Table A–13 UPLOAD Arguments

Argument	Definition
`-application` *application*	Specifies the name of the Oracle Event Processing application whose Oracle CQL rules you want to manage. To get the exact name of your application, you can: ▪ Use `wlevs.Admin` to query for the name (see Section , "Querying for Application and Processor Names"). ▪ Use the Oracle Event Processing Visualizer: Start the Oracle Event Processing Visualizer (see "Starting the Oracle Event Processing Visualizer" in the *Oracle Fusion Middleware Visualizer User's Guide for Oracle Event Processing*). In the left pane, navigate to and expand the **Applications** node of the Oracle Event Processing instance to which the application is deployed. Each node under the Applications node is named with the exact application name. ▪ Look at the `MANIFEST.MF` file of the application; the application name is specified by the `Bundle-SymbolicName` header.
`-processor` *processor*	Specifies the name of the particular Oracle CQL processor, attached to the Oracle Event Processing application specified with the `-application` argument, whose Oracle CQL rules you want to manage. See Section , "Querying for Application and Processor Names" for details on getting the exact name if you do not know it.
`-sourceURL` *sourcefileURL*	Specifies the URL of the component configuration file that contains the Oracle CQL rules in the form: `file:///`*path-to-file*

Example

The following example shows how upload the Oracle CQL rules in the `c:\processor\config\myrules.xml` file to the Oracle CQL `helloworldProcessor` of the `helloworld` application:

```
prompt> java wlevs.Admin
        -url service:jmx:msarmi://localhost:9002/jndi/jmxconnector
        -username wlevs -password wlevs
        UPLOAD -application helloworld -processor helloworldProcessor
        -sourceURL file:///c:/processor/config/myrules.xml
```

DOWNLOAD

Downloads the set of Oracle CQL rules associated with the specified Oracle CQL processor of an Oracle Event Processing application to an XML component configuration file.

The XML file is of the same format as described in Section , "UPLOAD."

Syntax

```
java wlevs.Admin
    [ Connection Arguments ]
    [ User Credentials Arguments ]
    [ Common Arguments ]
    DOWNLOAD -application application -processor processor
    -file destinationfile [-overwrite overwrite]
```

Table A–14 DOWNLOAD Arguments

Argument	Definition
`-application` *application*	Specifies the name of the Oracle Event Processing application whose Oracle CQL rules you want to manage. To get the exact name of your application, you can: ▪ Use `wlevs.Admin` to query for the name (see Section , "Querying for Application and Processor Names"). ▪ Use the Oracle Event Processing Visualizer: Start the Oracle Event Processing Visualizer (see "Starting the Oracle Event Processing Visualizer" in the *Oracle Fusion Middleware Visualizer User's Guide for Oracle Event Processing*). In the left pane, navigate to and expand the **Applications** node of the Oracle Event Processing instance to which the application is deployed. Each node under the Applications node is named with the exact application name. ▪ Look at the `MANIFEST.MF` file of the application; the application name is specified by the `Bundle-SymbolicName` header.
`-processor` *processor*	Specifies the name of the particular processor, attached to the Oracle Event Processing application specified with the `-application` argument, whose Oracle CQL rules you want to manage. See Section , "Querying for Application and Processor Names" for details on getting the exact name if you do not know it.
`-file` *destinationfile*	Specifies the name of the component configuration XML file to which you want the `wlevs.Admin` utility to download the Oracle CQL rules. Be sure you specify the full pathname of the file.
`-overwrite` *overwrite*	Specifies whether the `wlevs.Admin` utility should overwrite an existing file. Valid values for this argument are `true` or `false`; default value is `false`.

Example

The following example shows how download the set of Oracle CQL rules currently attached to the Oracle CQL `helloworldProcessor` of the `helloworld` application to the file `c:\processor\config\myrules.xml`; the utility overwrites any existing file:

```
prompt> java wlevs.Admin
        -url service:jmx:msarmi://localhost:9002/jndi/jmxconnector
        -username wlevs -password wlevs
        DOWNLOAD -application helloworld -processor helloworldProcessor
        -file c:\processor\config\myrules.xml -overwrite true
```

Commands for Managing the EPL Rules of an Application

Table A–15 is an overview of commands that manage the EPL rules for a particular processor of an Oracle Event Processing application. Subsequent sections describe command syntax and arguments, and provide an example for each command.

Table A–15 Overview of Commands for Managing Application EPL Rules

Command	Description
ADDRULE	Adds a new EPL rule to the processor of an Oracle Event Processing application.
DELETERULE	Deletes an existing EPL rule from the processor of an Oracle Event Processing application.
REPLACERULE	Replaces an existing EPL rule with new EPL text.
GETRULE	Returns the text of an existing EPL rule of the processor of an Oracle Event Processing application.
UPLOAD	Configures a set of EPL rules for a processor of an Oracle Event Processing application by uploading the rules from an XML file.

Table A–15 (Cont.) Overview of Commands for Managing Application EPL Rules

Command	Description
DOWNLOAD	Downloads the set of EPL rules associated with a processor of an Oracle Event Processing application to a file.
ADDPARAMS	Adds a new set of parameters to a parameterized EPL query.
DELETEPARAMS	Deletes a set of parameters from a parameterized EPL query.
GETPARAMS	Returns the parameters currently bound to a parameterized EPL query.

ADDRULE

Adds a new EPL rule to the specified processor of an Oracle Event Processing application.

If a rule with the same name (identified with the *rulename* parameter) already exists, then the ADDRULE command replaces the existing rule with the new one.

Syntax

```
java wlevs.Admin
    [ Connection Arguments ]
    [ User Credentials Arguments ]
    [ Common Arguments ]
    ADDRULE -application application -processor processor -rule [rulename] rulestring
```

Table A–16 ADDRULE Arguments

Argument	Definition
-application *application*	Specifies the name of the Oracle Event Processing application whose EPL rules you want to manage. To get the exact name of your application, you can: ■ Use wlevs.Admin to query for the name (see Section , "Querying for Application and Processor Names"). ■ Use the Oracle Event Processing Visualizer: Start the Oracle Event Processing Visualizer (see "Starting the Oracle Event Processing Visualizer" in the *Oracle Fusion Middleware Visualizer User's Guide for Oracle Event Processing*). In the left pane, navigate to and expand the **Applications** node of the Oracle Event Processing instance to which the application is deployed. Each node under the Applications node is named with the exact application name. ■ Look at the MANIFEST.MF file of the application; the application name is specified by the Bundle-SymbolicName header.
-processor *processor*	Specifies the name of the particular processor, attached to the Oracle Event Processing application specified with the -application argument, whose EPL rules you want to manage. See Section , "Querying for Application and Processor Names" for details on getting the exact name if you do not know it.
-rule [*rulename*] *rulestring*	Specifies the EPL rule you want to add to the specified processor of your application. The *rulename* parameter is not required; if you do not specify it, Oracle Event Processing generates a name for you. Enter the EPL rule using double quotes.

Example

The following example shows how to add the EPL rule SELECT * FROM Withdrawal RETAIN 5 EVENTS, with name myrule, to the helloworldProcessor of the helloworld application:

```
prompt> java wlevs.Admin
```

```
        -url service:jmx:msarmi://localhost:9002/jndi/jmxconnector
        -username wlevs -password wlevs
        ADDRULE -application helloworld -processor helloworldProcessor
        -rule myrule "SELECT * FROM Withdrawal RETAIN 5 EVENTS"
```

DELETERULE

Deletes an existing EPL rule from the specified processor of an Oracle Event Processing application.

Syntax

```
java wlevs.Admin
    [ Connection Arguments ]
    [ User Credentials Arguments ]
    [ Common Arguments ]
    DELETERULE -application application -processor processor -rule rulename
```

Table A–17 DELETERULE Arguments

Argument	Definition
`-application` *application*	Specifies the name of the Oracle Event Processing application whose EPL rules you want to manage. To get the exact name of your application, you can: ■ Use `wlevs.Admin` to query for the name (see Section , "Querying for Application and Processor Names"). ■ Use the Oracle Event Processing Visualizer: Start the Oracle Event Processing Visualizer (see "Starting the Oracle Event Processing Visualizer" in the *Oracle Fusion Middleware Visualizer User's Guide for Oracle Event Processing*). In the left pane, navigate to and expand the **Applications** node of the Oracle Event Processing instance to which the application is deployed. Each node under the Applications node is named with the exact application name. ■ Look at the `MANIFEST.MF` file of the application; the application name is specified by the `Bundle-SymbolicName` header.
`-processor` *processor*	Specifies the name of the particular processor, attached to the Oracle Event Processing application specified with the `-application` argument, whose EPL rules you want to manage. See Section , "Querying for Application and Processor Names" for details on getting the exact name if you do not know it.
`-rule` *rulename*	Specifies the name of the EPL rule you want to delete. See Section , "Querying for Application and Processor Names" for details on querying for the rule name if you do not know it. You can also use the DOWNLOAD command to get the list of rules for a particular processor.

Example

The following example shows how to delete the EPL rule called `myrule` from the `helloworldProcessor` of the `helloworld` application:

```
prompt> java wlevs.Admin
        -url service:jmx:msarmi://localhost:9002/jndi/jmxconnector
        -username wlevs -password wlevs
        DELETERULE -application helloworld -processor helloworldProcessor
-rule myrule
```

REPLACERULE

Replaces an existing EPL rule with another rule. Oracle Event Processing first destroys the original rule and then inserts the new one in its place. If the original rule was parameterized, any existing bindings are applied to the new rule.

Syntax

```
java wlevs.Admin
    [ Connection Arguments ]
    [ User Credentials Arguments ]
    [ Common Arguments ]
    REPLACERULE -application application -processor processor -rule rulename rulestring
```

Table A–18 REPLACERULE Arguments

Argument	Definition
`-application` *application*	Specifies the name of the Oracle Event Processing application whose EPL rules you want to manage. To get the exact name of your application, you can: ▪ Use `wlevs.Admin` to query for the name (see Section , "Querying for Application and Processor Names"). ▪ Use the Oracle Event Processing Visualizer: Start the Oracle Event Processing Visualizer (see "Starting the Oracle Event Processing Visualizer" in the *Oracle Fusion Middleware Visualizer User's Guide for Oracle Event Processing*). In the left pane, navigate to and expand the **Applications** node of the Oracle Event Processing instance to which the application is deployed. Each node under the Applications node is named with the exact application name. ▪ Look at the `MANIFEST.MF` file of the application; the application name is specified by the `Bundle-SymbolicName` header.
`-processor` *processor*	Specifies the name of the particular processor, attached to the Oracle Event Processing application specified with the `-application` argument, whose EPL rules you want to manage. See Section , "Querying for Application and Processor Names" for details on getting the exact name if you do not know it.
`-rule` *rulename rulestring*	Specifies the EPL rule you want to replace. Oracle Event Processing deletes the old rule and then inserts a new one, with the same name but with the new rule text. Enter the EPL rule using double quotes.

Example

The following example shows how to replace a rule called `myrule` with the EPL text `SELECT * FROM Withdrawal RETAIN 10 EVENTS` in the `helloworldProcessor` of the `helloworld` application:

```
prompt> java wlevs.Admin
        -url service:jmx:msarmi://localhost:9002/jndi/jmxconnector
        -username wlevs -password wlevs
        REPLACERULE -application helloworld -processor helloworldProcessor
        -rule myrule "SELECT * FROM Withdrawal RETAIN 10 EVENTS"
```

GETRULE

Returns the full text of an EPL rule from the specified processor of an Oracle Event Processing application.

Syntax

```
java wlevs.Admin
```

```
[ Connection Arguments ]
[ User Credentials Arguments ]
[ Common Arguments ]
GETRULE -application application -processor processor -rule rulename
```

Table A–19 GETRULE Arguments

Argument	Definition
`-application` *application*	Specifies the name of the Oracle Event Processing application whose EPL rules you want to manage. To get the exact name of your application, you can: ▪ Use `wlevs.Admin` to query for the name (see Section , "Querying for Application and Processor Names"). ▪ Use the Oracle Event Processing Visualizer: Start the Oracle Event Processing Visualizer (see "Starting the Oracle Event Processing Visualizer" in the *Oracle Fusion Middleware Visualizer User's Guide for Oracle Event Processing*). In the left pane, navigate to and expand the **Applications** node of the Oracle Event Processing instance to which the application is deployed. Each node under the Applications node is named with the exact application name. ▪ Look at the `MANIFEST.MF` file of the application; the application name is specified by the `Bundle-SymbolicName` header.
`-processor` *processor*	Specifies the name of the particular processor, attached to the Oracle Event Processing application specified with the `-application` argument, whose EPL rules you want to manage. See Section , "Querying for Application and Processor Names" for details on getting the exact name if you do not know it.
`-rule` *rulename*	Specifies the name of the EPL rule for which you want to view its full text. See Section , "Querying for Application and Processor Names" for details on querying for the rule name if you do not know it. You can also use the DOWNLOAD command to get the list of rules for a particular processor.

Example

The following example shows how to get the full text of the EPL rule called `myrule` from the `helloworldProcessor` of the `helloworld` application:

```
prompt> java wlevs.Admin
        -url service:jmx:msarmi://localhost:9002/jndi/jmxconnector
        -username wlevs -password wlevs
        GETRULE -application helloworld -processor helloworldProcessor
        -rule myrule
```

ADDPARAMS

Adds a new set of parameters to a parameterized EPL query.

See "Parameterized Queries" in the *Oracle Fusion Middleware EPL Language Reference for Oracle Event Processing* information about using parameterized EPL queries.

Syntax

```
java wlevs.Admin
    [ Connection Arguments ]
    [ User Credentials Arguments ]
    [ Common Arguments ]
    ADDPARAMS -application application -processor processor
    -rule rulename -values values -params params
```

Table A–20 ADDPARAMS Arguments

Argument	Definition
`-application application`	Specifies the name of the Oracle Event Processing application whose EPL rules you want to manage. To get the exact name of your application, you can: ▪ Use `wlevs.Admin` to query for the name (see Section , "Querying for Application and Processor Names"). ▪ Use the Oracle Event Processing Visualizer: Start the Oracle Event Processing Visualizer (see "Starting the Oracle Event Processing Visualizer" in the *Oracle Fusion Middleware Visualizer User's Guide for Oracle Event Processing*). In the left pane, navigate to and expand the **Applications** node of the Oracle Event Processing instance to which the application is deployed. Each node under the Applications node is named with the exact application name. ▪ Look at the `MANIFEST.MF` file of the application; the application name is specified by the `Bundle-SymbolicName` header.
`-processor processor`	Specifies the name of the particular processor, attached to the Oracle Event Processing application specified with the `-application` argument, whose parameterized EPL rules you want to manage. See Section , "Querying for Application and Processor Names" for details on getting the exact name if you do not know it.
`-rule rulename`	Specifies the name of the parameterized EPL rule for which you want add a new set of parameters. See Section , "Querying for Application and Processor Names" for details on querying for the rule name if you do not know it. You can also use the DOWNLOAD command to get the list of rules for a particular processor.
`-values values`	Specifies a comma-separated list of values that make up the parameter you want to add. Each value corresponds to a placeholder in the parameterized EPL query.
`-params params`	Specifies a unique identifier for this new parameter set.

Example

The following example shows how to use the ADDPARAMS command:

```
prompt> java wlevs.Admin
        -url service:jmx:msarmi://localhost:9002/jndi/jmxconnector
        -username wlevs -password wlevs
        ADDPARAMS -application myApplication -processor myProcessor
        -rule MarketRule
        -values "NYSE,BGP" -params nyBGP
```

The example shows how to add a parameter set identified by the string `nyBGP`, with values `NYSE,BGP`, to a parameterized query `MarketRule` running in the `myProcessor` component of `myApplication`. Because the parameter set is composed of two values, the EPL query must contain two placeholders.

DELETEPARAMS

Deletes one or all set of parameters associated with a parameterized EPL query.

See "Parameterized Queries" in the *Oracle Fusion Middleware EPL Language Reference for Oracle Event Processing* information about using parameterized EPL queries.

Syntax

```
java wlevs.Admin
    [ Connection Arguments ]
    [ User Credentials Arguments ]
    [ Common Arguments ]
    DELETEPARAMS -application application -processor processor
```

```
-rule rulename [-params params]
```

Table A–21 DELETEPARAMS Arguments

Argument	Definition
`-application application`	Specifies the name of the Oracle Event Processing application whose EPL rules you want to manage. To get the exact name of your application, you can: ▪ Use `wlevs.Admin` to query for the name (see Section , "Querying for Application and Processor Names"). ▪ Use the Oracle Event Processing Visualizer: Start the Oracle Event Processing Visualizer (see "Starting the Oracle Event Processing Visualizer" in the *Oracle Fusion Middleware Visualizer User's Guide for Oracle Event Processing*). In the left pane, navigate to and expand the **Applications** node of the Oracle Event Processing instance to which the application is deployed. Each node under the Applications node is named with the exact application name. ▪ Look at the `MANIFEST.MF` file of the application; the application name is specified by the `Bundle-SymbolicName` header.
`-processor processor`	Specifies the name of the particular processor, attached to the Oracle Event Processing application specified with the `-application` argument, whose parameterized EPL rules you want to manage. See Section , "Querying for Application and Processor Names" for details on getting the exact name if you do not know it.
`-rule rulename`	Specifies the name of the parameterized EPL rule for which you want to delete one or all of its parameter sets. See Section , "Querying for Application and Processor Names" for details on querying for the rule name if you do not know it. You can also use the DOWNLOAD command to get the list of rules for a particular processor.
`-params params`	Specifies the parameter set you want to delete. This argument is optional; if you do not specify it, `wlevs.Admin` deletes all parameter sets currently associated with the parameterized EPL rule.

Example

The following example shows how to use the DELETEPARAMS command:

```
prompt> java wlevs.Admin
        -url service:jmx:msarmi://localhost:9002/jndi/jmxconnector
        -username wlevs -password wlevs
        DELETEPARAMS -application myApplication -processor myProcessor
        -rule MarketRule
        -params nasORCL
```

The example shows how to delete the parameter set identified with the `nasORCL` string from the parameterized query `MarketRule` running in the `myProcessor` component of `myApplication`.

To delete all parameter sets associated to the query, do not specify the `-params` option:

```
prompt> java wlevs.Admin
-url service:jmx:msarmi://localhost:9002/jndi/jmxconnector
-username wlevs -password wlevs
DELETEPARAMS -application myApplication -processor myProcessor
-rule MarketRule
```

GETPARAMS

Returns one or all the parameter sets currently bound to a parameterized EPL query.

See "Parameterized Queries" in the *Oracle Fusion Middleware EPL Language Reference for Oracle Event Processing* information about using parameterized EPL queries.

Syntax

```
java wlevs.Admin
    [ Connection Arguments ]
    [ User Credentials Arguments ]
    [ Common Arguments ]
    GETPARAMS -application application -processor processor
    -rule rulename [-params params]
```

Table A–22 GETPARAMS Arguments

Argument	Definition
`-application application`	Specifies the name of the Oracle Event Processing application whose EPL rules you want to manage. To get the exact name of your application, you can: ▪ Use `wlevs.Admin` to query for the name (see Section , "Querying for Application and Processor Names"). ▪ Use the Oracle Event Processing Visualizer: Start the Oracle Event Processing Visualizer (see "Starting the Oracle Event Processing Visualizer" in the *Oracle Fusion Middleware Visualizer User's Guide for Oracle Event Processing*). In the left pane, navigate to and expand the **Applications** node of the Oracle Event Processing instance to which the application is deployed. Each node under the Applications node is named with the exact application name. ▪ Look at the `MANIFEST.MF` file of the application; the application name is specified by the `Bundle-SymbolicName` header.
`-processor processor`	Specifies the name of the particular processor, attached to the Oracle Event Processing application specified with the `-application` argument, whose parameterized EPL rules you want to manage. See Section , "Querying for Application and Processor Names" for details on getting the exact name if you do not know it.
`-rule rulename`	Specifies the name of the parameterized EPL rule for which you get the parameter sets. See Section , "Querying for Application and Processor Names" for details on querying for the rule name if you do not know it. You can also use the DOWNLOAD command to get the list of rules for a particular processor.
`-params params`	Specifies the parameter set you want return. This argument is optional; if you do not specify it, `wlevs.Admin` returns all parameter sets currently associated with the parameterized EPL rule.

Example

The following example shows how to use the GETPARAMS command:

```
prompt> java wlevs.Admin
        -url service:jmx:msarmi://localhost:9002/jndi/jmxconnector
        -username wlevs -password wlevs
        GETPARAMS -application myApplication -processor myProcessor
        -rule MarketRule
```

The example shows how to get all the parameters currently associated with the parameterized query `MarketRule` running in the `myProcessor` component of `myApplication`. The command would return something like:

```
    NASDAQ,ORCL
    NYSE,JPM
    NYSE,WFC
    NYSE,BGP
```

To retrieve a particular parameter set, specify its ID using the `-params` option:

```
prompt> java wlevs.Admin
        -url service:jmx:msarmi://localhost:9002/jndi/jmxconnector
```

```
-username wlevs -password wlevs
GETPARAMS -application myApplication -processor myProcessor
-rule MarketRule
-params nasORCL
```

UPLOAD

Replaces the configured EPL rules for a specified processor with the EPL rules from an uploaded XML file.

The XML file that contains the list of EPL rules conforms to component configuration file schema (see "Component Configuration XSD Schema: wlevs_application_config.xsd" in the *Oracle Fusion Middleware Developer's Guide for Oracle Event Processing for Eclipse*). This file contains one or more EPL rules that will replace those currently configured for the specified processor. An example of the XML file is as follows:

```
<?xml version="1.0" encoding="UTF-8" ?>
<config>
    <processor>
        <name>helloworldProcessor</name>
        <rules>
            <rule id="helloworldRule1">
                <![CDATA[  SELECT * FROM HelloWorldEvent RETAIN 2 EVENTS ]]>
            </rule>
        </rules>
    </processor>
</config>
```

In the preceding example, the XML file configures a single rule, with name `helloworldRule1`, and its EPL query text is `SELECT * FROM HelloWorldEvent RETAIN 2 EVENTS`.

> **Caution:** When you use the UPLOAD command of the `wlevs.Admin` utility, you use the -processor argument to specify the name of the processor to which you want to add the EPL rules, as you do with the other EPL commands. This means that the utility *ignores* any `<name>` elements in the XML file to avoid any naming conflicts.

See "Configuring EPL Processors" in the *Oracle Fusion Middleware Developer's Guide for Oracle Event Processing for Eclipse* for details and examples of creating the EPL rule XML file.

Syntax

```
java wlevs.Admin
    [ Connection Arguments ]
    [ User Credentials Arguments ]
    [ Common Arguments ]
    UPLOAD -application application -processor processor -sourceURL sourcefileURL
```

Table A–23 UPLOAD Arguments

Argument	Definition
`-application` *application*	Specifies the name of the Oracle Event Processing application whose EPL rules you want to manage. To get the exact name of your application, you can: ▪ Use `wlevs.Admin` to query for the name (see Section , "Querying for Application and Processor Names"). ▪ Use the Oracle Event Processing Visualizer: Start the Oracle Event Processing Visualizer (see "Starting the Oracle Event Processing Visualizer" in the *Oracle Fusion Middleware Visualizer User's Guide for Oracle Event Processing*). In the left pane, navigate to and expand the **Applications** node of the Oracle Event Processing instance to which the application is deployed. Each node under the Applications node is named with the exact application name. ▪ Look at the `MANIFEST.MF` file of the application; the application name is specified by the `Bundle-SymbolicName` header.
`-processor` *processor*	Specifies the name of the particular processor, attached to the Oracle Event Processing application specified with the `-application` argument, whose EPL rules you want to manage. See Section , "Querying for Application and Processor Names" for details on getting the exact name if you do not know it.
`-sourceURL` *sourcefileURL*	Specifies the URL of the XML file that contains the EPL rules.

Example

The following example shows how upload the EPL rules in the `c:\processor\config\myrules.xml` file to the `helloworldProcessor` of the `helloworld` application:

```
prompt> java wlevs.Admin
        -url service:jmx:msarmi://localhost:9002/jndi/jmxconnector
        -username wlevs -password wlevs
        UPLOAD -application helloworld -processor helloworldProcessor
        -sourceURL file:///c:/processor/config/myrules.xml
```

DOWNLOAD

Downloads the set of EPL rules associated with the specified processor of an Oracle Event Processing application to an XML file.

The XML file is of the same format as described in Section , "UPLOAD."

Syntax

```
java wlevs.Admin
    [ Connection Arguments ]
    [ User Credentials Arguments ]
    [ Common Arguments ]
    DOWNLOAD -application application -processor processor
    -file destinationfile [-overwrite overwrite]
```

Table A–24 DOWNLOAD Arguments

Argument	Definition
`-application` *application*	Specifies the name of the Oracle Event Processing application whose EPL rules you want to manage. To get the exact name of your application, you can: ■ Use `wlevs.Admin` to query for the name (see Section , "Querying for Application and Processor Names"). ■ Use the Oracle Event Processing Visualizer: Start the Oracle Event Processing Visualizer (see "Starting the Oracle Event Processing Visualizer" in the *Oracle Fusion Middleware Visualizer User's Guide for Oracle Event Processing*). In the left pane, navigate to and expand the **Applications** node of the Oracle Event Processing instance to which the application is deployed. Each node under the Applications node is named with the exact application name. ■ Look at the `MANIFEST.MF` file of the application; the application name is specified by the `Bundle-SymbolicName` header.
`-processor` *processor*	Specifies the name of the particular processor, attached to the Oracle Event Processing application specified with the `-application` argument, whose EPL rules you want to manage. See Section , "Querying for Application and Processor Names" for details on getting the exact name if you do not know it.
`-file` *destinationfile*	Specifies the name of the XML file to which you want the wlevs.Admin utility to download the EPL rules. Be sure you specify the full pathname of the file.
`-overwrite` *overwrite*	Specifies whether the `wlevs.Admin` utility should overwrite an existing file. Valid values for this argument are `true` or `false`; default value is `false`.

Example

The following example shows how download the set of EPL rules currently attached to the `helloworldProcessor` of the `helloworld` application to the file `c:\processor\config\myrules.xml`; the utility overwrites any existing file:

```
prompt> java wlevs.Admin
        -url service:jmx:msarmi://localhost:9002/jndi/jmxconnector
        -username wlevs -password wlevs
        DOWNLOAD -application helloworld -processor helloworldProcessor
        -file c:\processor\config\myrules.xml
```

Commands for Managing Oracle Event Processing MBeans

The following sections describe `wlevs.Admin` commands for managing Oracle Event Processing MBeans.

- Section , "Specifying MBean Types"
- Section , "MBean Management Commands"

See the *Oracle Fusion Middleware Java API Reference for Oracle Event Processing* for the full description of the Oracle Event Processing MBeans.

Specifying MBean Types

To specify which MBean or MBeans you want to access, view, or modify, all of the MBean management commands require either the `-mbean` argument or the `-type` argument.

Use the `-mbean` argument to operate on a single instance of an MBean.

Use the -type argument to operate on all MBeans that are an instance of a type that you specify. An MBean's type refers to the interface class of which the MBean is an instance. All Oracle Event Processing MBeans are an instance of one of the interface classes defined in the com.bea.wlevs.management.configuration, com.bea.wlevs.management.runtime, com.bea.wlevs.deployment.mbean and com.bea.wlevs.server.management.mbean packages. For a complete list of all Oracle Event Processing MBean interface classes, see the *Oracle Fusion Middleware Java API Reference for Oracle Event Processing* for the respective packages.

To determine the value that you provide for the -type argument, do the following: Find the MBean's interface class and remove the MBean suffix from the class name. For example, for an MBean that is an instance of the com.bea.wlevs.management.configuration.CQLProcessorMBean, use CQLProcessor.

MBean Management Commands

Table A–25 is an overview of the MBean management commands.

Table A–25 MBean Management Command Overview

Command	Description
GET	Displays properties of MBeans.
INVOKE	Invokes management operations that an MBean exposes for its underlying resource.
QUERY	Searches for MBeans whose ObjectName matches a pattern that you specify.
SET	Sets the specified property values for the named MBean instance.

GET

Displays MBean properties (attributes) and JMX object names (in the http://java.sun.com/j2se/1.5.0/docs/api/javax/management/ObjectName.html format).

The output of the command is as follows:

```
{MBeanName object-name {property1 value} {property2 value}. . .}
. . .
```

Note that the properties and values are expressed as name-value pairs, each of which is returned within curly brackets. This format facilitates parsing of the output by a script.

If -pretty is specified, each property-value pair is displayed on a new line and curly brackets are not used to separate the pairs:

```
MBeanName: object-name
property1: value
property2: value
.
.
.
MBeanName: object-name
property1: value
abbribute2: value
```

Syntax

```
java wlevs.Admin
    [ Connection Arguments ]
    [ User Credentials Arguments ]
    [ Common Arguments ]
```

```
    GET [-pretty] {-type mbeanType| -mbean objectName} [-property property1] [-property
property2]...
```

Table A–26 GET Arguments

Argument	Definition
`-type mbeanType`	Returns information for all MBeans of the specified type. For more information, see Section , "Specifying MBean Types."
`-mbean objectName`	Fully qualified object name of an MBean in the `http://java.sun.com/j2se/1.5.0/docs/api/javax/management/ObjectName.html` format. For example, if you want to look up an MBean for an EPL Processor Stage, the naming is as follows (in practice, the string should be on one line): `"com.bea.wlevs:Name=<name of the Stage>,Type=<type of Mbean>, Application=<name of the application>"`
`-pretty`	Places property-value pairs on separate lines.
`-property property`	The name of the MBean property (attribute) or properties to be listed. If `-property` is not specified, all properties are displayed.

Example

The following example displays all properties of the `CQLProcessorMBean` that was registered for the Processor Stage when the application called `helloworld` was deployed in Oracle Event Processing.

```
prompt> java wlevs.Admin
        -url service:jmx:msarmi://localhost:9002/jndi/jmxconnector
        -username wlevs -password wlevs
        GET -pretty
        -mbean com.bea.wlevs:Name=eplprocessor,Type=CQLProcessor,Application=helloworld
```

The following example displays all instances of all `CQLProcessorMBean` MBeans.

```
prompt> java wlevs.Admin
        -url service:jmx:msarmi://localhost:9002/jndi/jmxconnector
        -username wlevs -password wlevs
        GET -pretty -type CQLProcessor
```

INVOKE

Invokes a management operation for one or more MBeans. For Oracle Event Processing MBeans, you usually use this command to invoke operations other than the `get`*`Attribute`* and `set`*`Attribute`* that most Oracle Event Processing MBeans provide.

Syntax

```
java wlevs.Admin
    [ Connection Arguments ]
    [ User Credentials Arguments ]
    [ Common Arguments ]
    INVOKE {-type mbeanType | -mbean objectName} -method methodname [argument . . .]
```

Table A–27 INVOKE Arguments

Arguments	Definition
`-type mbeanType`	Invokes the operation on all MBeans of a specific type. For more information, see Section , "Specifying MBean Types."

Table A–27 (Cont.) INVOKE Arguments

Arguments	Definition
-mbean *objectName*	Fully qualified object name of an MBean in the http://java.sun.com/j2se/1.5.0/docs/api/javax/management/ObjectName.html format. For example, if you want to invoke an MBean for an EPL Processor Stage, the naming is as follows "com.bea.wlevs:Name=<name of the Stage>,Type=<type of Mbean>, Application=<name of the application>"
-method *methodname*	Name of the method to be invoked.
argument	Arguments to be passed to the method call. When the argument is a String array, the arguments must be passed in the following format: "*String1*;*String2*;. . . "

Example

The following example invokes the `addRule` method of MBean `com.bea.wlevs.management.configuration.EPLProcessorMBean`:

```
prompt> java wlevs.Admin
        -url service:jmx:msarmi://localhost:9002/jndi/jmxconnector
        -username wlevs -password wlevs
        INVOKE -mbean
com.bea.wlevs:Name=eplprocessor,Type=EPLProcessor,Application=helloworld
        -method addRule "SELECT * FROM Withdrawal RETAIN ALL"
```

QUERY

Searches for Oracle Event Processing MBeans whose http://java.sun.com/j2se/1.5.0/docs/api/javax/management/ObjectName.html matches a pattern that you specify.

All MBeans that are created from an Oracle Event Processing MBean type are registered in the MBean Server under a name that conforms to the http://java.sun.com/j2se/1.5.0/docs/api/javax/management/ObjectName.html conventions. You must know an MBean's `ObjectName` if you want to use `wlevs.Admin` commands to retrieve or modify specific MBean instances.

The output of the command is as follows:

```
{MBeanName object-name {property1 value} {property2 value}. . .}
. . .
```

Note that the properties and values are expressed as name-value pairs, each of which is returned within curly brackets. This format facilitates parsing of the output by a script.

If `-pretty` is specified, each property-value pair is displayed on a new line and curly brackets are not used to separate the pairs:

```
MBeanName: object-name
property1: value
property2: value
.
.
.
MBeanName: object-name
property1: value
abbribute2: value
```

Syntax

```
java wlevs.Admin
    [ Connection Arguments ]
    [ User Credentials Arguments ]
    [ Common Arguments ]
    QUERY -pretty -pattern object-name-pattern
```

Table A–28 QUERY Arguments

Argument	Definition
`-pretty`	Places property-value pairs on separate lines.
`-pattern` *`object-name-pattern`*	A partial `http://java.sun.com/j2se/1.5.0/docs/api/javax/management/ObjectName.html` for which the `QUERY` command searches. The value must conform to the following pattern: *`property-list`* where property-list specifies one or more components (property-value pairs) of a `http://java.sun.com/j2se/1.5.0/docs/api/javax/management/ObjectName.html`. You can specify these property-value pairs in any order. Within a given naming property-value pair, there is no pattern matching. Only complete property-value pairs are used in pattern matching. However, you can use the * wildcard character in the place of one or more property-value pairs. For example, `type=epl*` is not valid, but `type=EPLProcessor,*` is valid. If you provide at least one property-value pair in the *`property-list`*, you can locate the wildcard anywhere in the given pattern, provided that the *`property-list`* is still a comma-separated list.

Example

The following example searches for all `com.bea.wlevs.management.configuration.EPLProcessorMBean` MBeans:

```
prompt> java wlevs.Admin
        -url service:jmx:msarmi://localhost:9002/jndi/jmxconnector
        -username wlevs -password wlevs
        QUERY -pattern *:Type=EPLProcessor,*
```

If the command succeeds, it returns the attributes of the MBeans found (lines broken here for readability):

```
{MBeanName="com.bea.wlevs:Name=MonitorProcessor,Type=EPLProcessor,Application=com.bea.wlevs.d
ataservices"{
    AllRules=defaultRule = select * from DSMonitorEvent retain 1 event where metric > 10000}{
    AllRulesInfo=defaultRule = {RULE_TYPE=RULE, STARTED=true, VALUE=select * from
DSMonitorEvent
    retain 1 event where metric > 10000, ID=defaultRule}}{Databases=}{Name=MonitorProcessor}
    {NotificationInfo=[Ljavax.management.MBeanNotificationInfo;@20d319}
    {ObjectName=com.bea.wlevs:Name=MonitorProcessor,Type=EPLProcessor,
    Application=com.bea.wlevs.dataservices}{PlaybackConfiguration=}{PlayingBack=false}
    {RecordConfiguration=}{RecordPlayback=com.bea.wlevs:Name=MonitorProcessor,
    Type=RecordPlayback,Application=com.bea.wlevs.dataservices}{Recording=false}
    {Type=EPLProcessor}}
```

Querying for Application and Processor Names

All the commands for managing the EPL rules of an Oracle Event Processing application require you know the name of the application, as well the particular processor to which you want to apply the rules. Typically you know these names, but

if you do not, you can use the QUERY command to get the information from the MBean instances that represent applications and their attached processors.

In particular, use the following -pattern argument to get a list of all applications, processors, and rules for a given Oracle Event Processing instance:

```
-pattern  com.bea.wlevs:*,Type=EPLProcessor
```

For example:

```
prompt> java wlevs.Admin -url
    service:jmx:msarmi://localhost:9002/jndi/jmxconnector
    -username wlevs -password wlevs
    QUERY -pretty
    -pattern com.bea.wlevs:*,Type=EPLProcessor
```

A sample output of this command is shown below:

```
Command Output
------------------------------------------------------
MBeanName: "com.bea.wlevs:Name=helloworldProcessor,Type=EPLProcessor,Application=helloworld,"
       AllRules:
       helloworldRule =  select * from HelloWorldEvent retain 1 event
--end of command output --------
```

In the sample output above:

- The name of the application is helloworld.
- The helloworld application has a processor called helloworldProcessor.
- The helloworldProcessor has a rule called helloworldRule.

SET

Sets the specified property (attribute) values for an MBean.

If the command is successful, it returns OK and saves the new values to the server configuration.

Syntax

```
java wlevs.Admin
    [ Connection Arguments ]
    [ User Credentials Arguments ]
    [ Common Arguments ]
    SET {-type mbeanType | -mbean objectName}
    -property property1 property1_value
    [-property property2 property2_value] . . .
```

Table A–29 SET Arguments

Argument	Definition
-type *mbeanType*	Sets the properties for all MBeans of a specific type. For more information, see Section , "Specifying MBean Types."
-mbean *objectName*	Fully qualified object name of an MBean in the http://java.sun.com/j2se/1.5.0/docs/api/javax/management/ObjectName.html format: "com.bea.wlevs:Name=<name of the stage>,Type=<MBean type>,Application=<name of the deployed application>"
-property *property*	The name of the property to be set.

Table A–29 (Cont.) SET Arguments

Argument	Definition

property _value	The value to be set.
	■ Some properties require you to specify the name of an Oracle Event Processing MBean. In this case, specify the fully qualified object name of an MBean in the http://java.sun.com/j2se/1.5.0/docs/api/javax/management/ObjectName.html format. For example (in practice, the string should be on one line):
	`"com.bea.wlevs:Name=<name of the stage>,Type=<type of MBean>, Application=<name of the application>"`
	■ When the property value is an MBean array, separate each MBean object name by a semicolon and surround the entire property value list with quotes. For example:
	`"com.bea.wlevs:Application=<name of the application>,Type=<type of MBean>,Name=<name of the Stage>;Type=<type of MBean>,Name=<name of the stage>"`
	■ When the property value is a String array, separate each string by a semicolon and surround the entire property value list with quotes:
	`"String1;String2;. . . "`
	■ When the property value is a String or String array, you can set the value to null by using either of the following:
	`-property property-name ""`
	`-property property-name`
	■ If the property value contains spaces, surround the value with quotes:
	`"-Da=1 -Db=3"`

Example

The following example shows how to set the `MaxSize` property of the channel named `helloworldOutstream` of the `helloworld` application:

```
prompt> java wlevs.Admin
        -url service:jmx:msarmi://localhost:9002/jndi/jmxconnector
        -username wlevs -password wlevs
        SET -mbean com.bea.wlevs:Name=helloworldOutstream,Type=Channel,Application=helloworld
        -property  MaxSize 1024
```

Commands for Controlling Event Record and Playback

Table A–30 is an overview of commands for managing event record and playback for a particular stage of an Oracle Event Processing application. Subsequent sections describe command syntax and arguments, and provide an example for each command.

Note: Before you can use commands for controlling event record and playback on a stage, you must first configure the stage with the appropriate event record and playback options. For more information, see:

- Section , "CONFIGURERECORD"
- "Configuring Event Record and Playback" in the *Oracle Fusion Middleware Developer's Guide for Oracle Event Processing for Eclipse*

Table A–30 Overview of Commands for Controlling Event Record and Playback

Command	Description
STARTRECORD	Starts the recording of events for a stage in an Oracle Event Processing application.
STOPRECORD	Stops the recording of events for a stage in an Oracle Event Processing application.
CONFIGURERECORD	Configures the parameters for the event recording of a stage in an Oracle Event Processing application.
SCHEDULERECORD	Schedules the recording of events for a stage in an Oracle Event Processing application.
LISTRECORD	List the event recording configuration for a stage in an Oracle Event Processing application.
STARTPLAYBACK	Starts playing back events for a stage in an Oracle Event Processing application.
STOPPLAYBACK	Stops playing back events for a stage in an Oracle Event Processing application.
CONFIGUREPLAYBACK	Configures the parameters for the event playback of a stage in an Oracle Event Processing application.
SCHEDULEPLAYBACK	Schedules the playback of events for a stage in an Oracle Event Processing application.
LISTPLAYBACK	List the event playback configuration for a stage in an Oracle Event Processing application.

STARTRECORD

Starts the recording of events for any particular stage of an Oracle Event Processing application.

Note: Before you can use commands for controlling event record and playback on a stage, you must first configure the stage with the appropriate event record and playback options. For more information, see:

- Section , "CONFIGURERECORD"
- "Configuring Event Record and Playback" in the *Oracle Fusion Middleware Developer's Guide for Oracle Event Processing for Eclipse*

If you configured the stage to start recording at a later time, that configuration is ignored and recording starts immediately.

Syntax

```
java wlevs.Admin
    [ Connection Arguments ]
    [ User Credentials Arguments ]
    [ Common Arguments ]
    STARTRECORD -application application -stage stage
```

Table A–31 STARTRECORD Arguments

Argument	Definition
`-application` *application*	Specifies the name of the Oracle Event Processing application whose event record and playback you want to manage. To get the exact name of your application, you can: ■ Use `wlevs.Admin` to query for the name (see Section , "Querying for Application and Processor Names"). ■ Use the Oracle Event Processing Visualizer: Start the Oracle Event Processing Visualizer (see "Starting the Oracle Event Processing Visualizer" in the *Oracle Fusion Middleware Visualizer User's Guide for Oracle Event Processing*). In the left pane, navigate to and expand the **Applications** node of the Oracle Event Processing instance to which the application is deployed. Each node under the Applications node is named with the exact application name. ■ Look at the `MANIFEST.MF` file of the application; the application name is specified by the `Bundle-SymbolicName` header.
`-stage` *stage*	Specifies the name of the particular stage, attached to the Oracle Event Processing application specified with the `-application` argument, whose event record and playback you want to manage.

Example

The following example shows how to start the recording of events on the `helloworldAdapter` stage of the `helloworld` application deployed to the specified Oracle Event Processing instance:

```
prompt> java wlevs.Admin
    -url service:jmx:msarmi://localhost:9002/jndi/jmxconnector
    -username wlevs -password wlevs
    STARTRECORD -application helloworld -stage helloworldAdapter
```

Note: Before you can use commands for controlling event record and playback on a stage, you must first configure the stage with the appropriate event record and playback options. For more information, see:

- Section , "CONFIGURERECORD"
- "Configuring Event Record and Playback" in the *Oracle Fusion Middleware Developer's Guide for Oracle Event Processing for Eclipse*

STOPRECORD

Stops the recording of events for a stage of an Oracle Event Processing application in which the recording of events has been previously started.

Syntax

```
java wlevs.Admin
    [ Connection Arguments ]
    [ User Credentials Arguments ]
    [ Common Arguments ]
    STOPRECORD -application application -stage stage
```

Table A–32 STOPRECORD Arguments

Argument	Definition
`-application application`	Specifies the name of the Oracle Event Processing application whose event record and playback you want to manage. To get the exact name of your application, you can: ▪ Use `wlevs.Admin` to query for the name (see Section , "Querying for Application and Processor Names"). ▪ Use the Oracle Event Processing Visualizer: Start the Oracle Event Processing Visualizer (see "Starting the Oracle Event Processing Visualizer" in the *Oracle Fusion Middleware Visualizer User's Guide for Oracle Event Processing*). In the left pane, navigate to and expand the **Applications** node of the Oracle Event Processing instance to which the application is deployed. Each node under the Applications node is named with the exact application name. ▪ Look at the `MANIFEST.MF` file of the application; the application name is specified by the `Bundle-SymbolicName` header.
`-stage stage`	Specifies the name of the particular stage, attached to the Oracle Event Processing application specified with the `-application` argument, whose event record and playback you want to manage.

Example

The following example shows how to stop the recording of events on the `helloworldAdapter` stage of the `helloworld` application deployed to the specified Oracle Event Processing instance; it is assumed that the recording of events was previously started for the stage:

```
prompt> java wlevs.Admin
    -url service:jmx:msarmi://localhost:9002/jndi/jmxconnector
    -username wlevs -password wlevs
    STOPRECORD -application helloworld -stage helloworldAdapter
```

CONFIGURERECORD

Configures the parameters associated with the recording of events for a stage of an Oracle Event Processing application.

You typically use this command to configure a stage for the first time for event recording or to change the dataset name or provider name. For more information, see "Configuring Event Record and Playback" in the *Oracle Fusion Middleware Developer's Guide for Oracle Event Processing for Eclipse*.

Syntax

```
java wlevs.Admin
    [ Connection Arguments ]
    [ User Credentials Arguments ]
    [ Common Arguments ]
    CONFIGURERECORD -application application -stage stage
        [-datasetName datasetname]
        [-storeProvider storeprovidername]
        [-eventTypes eventtypes]
        [-scheduleStartTime startime]
        [-scheduleEndTime endtime | -scheduleDuration duration]
```

Table A–33 CONFIGURERECORD Arguments

Argument	Definition
`-application` *application*	Specifies the name of the Oracle Event Processing application whose event record and playback you want to manage. To get the exact name of your application, you can: ■ Use `wlevs.Admin` to query for the name (see Section , "Querying for Application and Processor Names"). ■ Use the Oracle Event Processing Visualizer: Start the Oracle Event Processing Visualizer (see "Starting the Oracle Event Processing Visualizer" in the *Oracle Fusion Middleware Visualizer User's Guide for Oracle Event Processing*). In the left pane, navigate to and expand the **Applications** node of the Oracle Event Processing instance to which the application is deployed. Each node under the Applications node is named with the exact application name. ■ Look at the `MANIFEST.MF` file of the application; the application name is specified by the `Bundle-SymbolicName` header.
`-stage` *stage*	Specifies the name of the particular stage, attached to the Oracle Event Processing application specified with the `-application` argument, whose event record and playback you want to manage.
`-datasetName` *datasetname*	Specifies the name of the dataset in which events are recorded.
`-storeProvider` *storeprovidername*	Specifies a valid `data-source` name defined in the Oracle Event Processing server `config.xml` file. To select the default BDB provider, leave this argument empty or specify an argument value of `default-provider`. For more information, see "Configuring an Event Store for Oracle Event Processing Server" in the *Oracle Fusion Middleware Developer's Guide for Oracle Event Processing for Eclipse*.
`-eventTypes` *eventtypes*	Specifies the comma-separated list of valid event type names to be recorded. Event types must be defined in the event type repository.

`-scheduleStartTime` *starttime*	Specifies the time when the recording should start.
	Express the start time as an XMLSchema `dateTime` value of the form:
	`yyyy-mm-ddThh:mm:ss`
	For example, to specify that recording should start on January 20, 2010, at 5:00am, use the following value:
	`2010-01-20T05:00:00`
	For complete details of the XMLSchema `dateTime` format, see `http://www.w3.org/TR/xmlschema-2/#dateTime-lexical-representation`.

`-scheduleEndTime` *endtime*	Specifies the actual time when the recording should end.
	Express the end time as an XMLSchema `dateTime` value of the form:
	`yyyy-mm-ddThh:mm:ss`
	For example, to specify that recording should end on January 20, 2010, at 6:00pm, use the following value:
	`2010-01-20T18:00:00`
	For complete details of the XMLSchema `dateTime` format, see `http://www.w3.org/TR/xmlschema-2/#dateTime-lexical-representation`.
	Specify `null` if you want the recording to run forever.
	You can specify either `-scheduleEndTime` or `-scheduleDuration`, but not both.

Table A–33 (Cont.) CONFIGURERECORD Arguments

Argument	Definition
-scheduleDuration *duration*	Specifies the duration of time after which event recording for this stage ends. Specify null if you want the recording to run forever. The format is HH:mm:ss, such as 01:00:00. You can specify either -scheduleEndTime or -scheduleDuration, but not both.

Example

The examples in this section show how to configure the recording of events of the helloworldAdapter of the helloworld application deployed to the specified Oracle Event Processing instance.

The following example specifies a start and end time for recording:

```
prompt> java wlevs.Admin
        -url service:jmx:msarmi://localhost:9002/jndi/jmxconnector
        -username wlevs -password wlevs
        CONFIGURERECORD -application helloworld -stage helloworldAdapter
        -datasetName myds -storeProvider mysp
        -scheduleStartTime 2010-01-20T05:00:00 -scheduleEndndTime 2010-01-20T18:00:00
```

The following example specifies a start time and a duration for recording:

```
prompt> java wlevs.Admin
        -url service:jmx:msarmi://localhost:9002/jndi/jmxconnector
        -username wlevs -password wlevs
        CONFIGURERECORD -application helloworld -stage helloworldAdapter
        -datasetName myds -storeProvider mysp
        -scheduleStartTime 2010-01-20T05:00:00 -scheduleDuration 01:00:00
```

The following example specifies a start time and a duration of null, which means recording will run forever:

```
prompt> java wlevs.Admin
        -url service:jmx:msarmi://localhost:9002/jndi/jmxconnector
        -username wlevs -password wlevs
        CONFIGURERECORD -application helloworld -stage helloworldAdapter
        -datasetName myds -storeProvider mysp
        -scheduleStartTime 2010-01-20T05:00:00 -scheduleDuration null
```

SCHEDULERECORD

Configures the schedule parameters associated with the recording of events for a stage of an Oracle Event Processing application.

Syntax

```
java wlevs.Admin
    [ Connection Arguments ]
    [ User Credentials Arguments ]
    [ Common Arguments ]
    SCHEDULERECORD -application application -stage stage
        [-scheduleStartTime startime]
        [-scheduleEndTime endtime | -scheduleDuration duration]
```

Table A–34 SCHEDULERECORD Arguments

Argument	Definition
`-application` *application*	Specifies the name of the Oracle Event Processing application whose event record and playback you want to manage. To get the exact name of your application, you can: ▪ Use `wlevs.Admin` to query for the name (see Section , "Querying for Application and Processor Names"). ▪ Use the Oracle Event Processing Visualizer: Start the Oracle Event Processing Visualizer (see "Starting the Oracle Event Processing Visualizer" in the *Oracle Fusion Middleware Visualizer User's Guide for Oracle Event Processing*). In the left pane, navigate to and expand the **Applications** node of the Oracle Event Processing instance to which the application is deployed. Each node under the Applications node is named with the exact application name. ▪ Look at the `MANIFEST.MF` file of the application; the application name is specified by the `Bundle-SymbolicName` header.
`-stage` *stage*	Specifies the name of the particular stage, attached to the Oracle Event Processing application specified with the `-application` argument, whose event record and playback you want to manage.

`-scheduleStartTime` *starttime*	Specifies the time when the recording should start.
	Express the start time as an XMLSchema `dateTime` value of the form:
	`yyyy-mm-ddThh:mm:ss`
	For example, to specify that recording should start on January 20, 2010, at 5:00am, use the following value:
	`2010-01-20T05:00:00`
	For complete details of the XMLSchema `dateTime` format, see http://www.w3.org/TR/xmlschema-2/#dateTime-lexical-representation.

`-scheduleEndTime` *endtime*	Specifies the actual time when the recording should end.
	Express the end time as an XMLSchema `dateTime` value of the form:
	`yyyy-mm-ddThh:mm:ss`
	For example, to specify that recording should end on January 20, 2010, at 6:00pm, use the following value:
	`2010-01-20T18:00:00`
	For complete details of the XMLSchema `dateTime` format, see http://www.w3.org/TR/xmlschema-2/#dateTime-lexical-representation.
	Specify `null` if you want the recording to run forever.
	You can specify either `-scheduleEndTime` or `-scheduleDuration`, but not both.

`-scheduleDuration` *duration*	Specifies the duration of time after which event recording for this stage ends. Specify `null` if you want the recording to run forever. The format is `HH:mm:ss`, such as `01:00:00`. You can specify either `-scheduleEndTime` or `-scheduleDuration`, but not both.

Example

The examples in this section show how to configure the scheduling of recording of events of the `helloworldAdapter` of the `helloworld` application deployed to the specified Oracle Event Processing instance.

The following example specifies a start and end time for recording:

```
prompt> java wlevs.Admin
```

```
    -url service:jmx:msarmi://localhost:9002/jndi/jmxconnector
    -username wlevs -password wlevs
    SCHEDULERECORD -application helloworld -stage helloworldAdapter
    -scheduleStartTime 2010-01-20T05:00:00 -scheduleEndndTime 2010-01-20T18:00:00
```

LISTRECORD

Lists the event recording configuration for any particular stage of an Oracle Event Processing application.

Syntax

```
java wlevs.Admin
    [ Connection Arguments ]
    [ User Credentials Arguments ]
    [ Common Arguments ]
    LISTRECORD -application application -stage stage
```

Table A–35 LISTRECORD Arguments

Argument	Definition
`-application` *application*	Specifies the name of the Oracle Event Processing application whose event record and playback you want to manage. To get the exact name of your application, you can: ▪ Use `wlevs.Admin` to query for the name (see Section , "Querying for Application and Processor Names"). ▪ Use the Oracle Event Processing Visualizer: Start the Oracle Event Processing Visualizer (see "Starting the Oracle Event Processing Visualizer" in the *Oracle Fusion Middleware Visualizer User's Guide for Oracle Event Processing*). In the left pane, navigate to and expand the **Applications** node of the Oracle Event Processing instance to which the application is deployed. Each node under the Applications node is named with the exact application name. ▪ Look at the `MANIFEST.MF` file of the application; the application name is specified by the `Bundle-SymbolicName` header.
`-stage` *stage*	Specifies the name of the particular stage, attached to the Oracle Event Processing application specified with the `-application` argument, whose event record and playback you want to manage.

Example

The following example shows how to list the event recording configuration on the `helloworldAdapter` stage of the `helloworld` application deployed to the specified Oracle Event Processing instance:

```
prompt> java wlevs.Admin
    -url service:jmx:msarmi://localhost:9002/jndi/jmxconnector
    -username wlevs -password wlevs
    LISTRECORD -application helloworld -stage helloworldAdapter
```

STARTPLAYBACK

Starts the playback of events of a particular stage of a Oracle Event Processing application.

Syntax

```
java wlevs.Admin
    [ Connection Arguments ]
    [ User Credentials Arguments ]
```

```
[ Common Arguments ]
STARTPLAYBACK -application application -stage stage
    [-filterStartTime starttime]
    [-filterEndTime endtime | -filterDuration duration]        [-speed speed]
    [-repeat true | false]
```

Table A–36 STARTPLAYBACK Arguments

Argument	Definition
`-application` *application*	Specifies the name of the Oracle Event Processing application whose event record and playback you want to manage. To get the exact name of your application, you can: ▪ Use `wlevs.Admin` to query for the name (see Section , "Querying for Application and Processor Names"). ▪ Use the Oracle Event Processing Visualizer: Start the Oracle Event Processing Visualizer (see "Starting the Oracle Event Processing Visualizer" in the *Oracle Fusion Middleware Visualizer User's Guide for Oracle Event Processing*). In the left pane, navigate to and expand the **Applications** node of the Oracle Event Processing instance to which the application is deployed. Each node under the Applications node is named with the exact application name. ▪ Look at the `MANIFEST.MF` file of the application; the application name is specified by the `Bundle-SymbolicName` header.
`-stage stage`	Specifies the name of the particular stage, attached to the Oracle Event Processing application specified with the `-application` argument, whose event record and playback you want to manage.

`-filterStartTime` *startime*	Specifies that only events with record-time greater than or equal to this value will be selected for playback.
	Express the start time as an XMLSchema `dateTime` value of the form:
	`yyyy-mm-ddThh:mm:ss`
	For example, to play back only events with record-time greater than or equal to January 20, 2010, at 5:00am, use the following value:
	`2010-01-20T05:00:00`
	For complete details of the XMLSchema `dateTime` format, see http://www.w3.org/TR/xmlschema-2/#dateTime-lexical-representation.
	Specify `null` if you want to select all events for playback.

`-filterEndTime` *endtime*	Specifies only events with record-time less than or equal to this value will be selected for playback.
	Express the end time as an XMLSchema `dateTime` value of the form:
	`yyyy-mm-ddThh:mm:ss`
	For example, to play back only events with record-time less than or equal to January 20, 2010, at 6:00pm, use the following value:
	`2010-01-20T18:00:00`
	For complete details of the XMLSchema `dateTime` format, see http://www.w3.org/TR/xmlschema-2/#dateTime-lexical-representation.
	You can specify either `-filterEndTime` or `-filterDuration`, but not both.

`-filterDuration` *duration*	Specifies the filter applied to events in the event store. Only events that were recorded during the filter time will be selected for play back. Specify `null` if you want to select all events for playback. The format is `HH:mm:ss`, such as `01:00:00`. You can specify either `-filterEndTime` or `-filterDuration`, but not both.

Table A–36 (Cont.) STARTPLAYBACK Arguments

Argument	Definition
`-speed` *speed*	Specifies the playback speed as a positive float. The default value is 1, which corresponds to normal speed. A value of 2 means that events will be played back 2 times faster than the original record speed. Similarly, a value of 0.5 means that events will be played back 2 times slower than the original record speed.
`-repeat` *repeat*	Specifies whether to playback events again after the playback of the specified time interval is over. Valid values are `true` or `false`. Default value is `false`. A value of `true` means that the repeat of playback continues an infinite number of times until it is deliberately stopped (see STOPPLAYBACK). A value of `false` means that events will be played back only once.

Example

The following example shows how to start the playback of events on the `helloworldAdapter` stage of the `helloworld` application deployed to the specified Oracle Event Processing instance:

```
prompt> java wlevs.Admin
    -url service:jmx:msarmi://localhost:9002/jndi/jmxconnector
    -username wlevs -password wlevs
    STARTPLAYBACK -application helloworld -stage helloworldAdapter
```

STOPPLAYBACK

Stops the playback of events for a stage of an Oracle Event Processing application in which the playback of events has been previously started.

Syntax

```
java wlevs.Admin
    [ Connection Arguments ]
    [ User Credentials Arguments ]
    [ Common Arguments ]
    STOPPLAYBACK -application application -stage stage
```

Table A–37 STOPPLAYBACK Arguments

Argument	Definition
`-application` *application*	Specifies the name of the Oracle Event Processing application whose event record and playback you want to manage. To get the exact name of your application, you can: ▪ Use `wlevs.Admin` to query for the name (see Section , "Querying for Application and Processor Names"). ▪ Use the Oracle Event Processing Visualizer: Start the Oracle Event Processing Visualizer (see "Starting the Oracle Event Processing Visualizer" in the *Oracle Fusion Middleware Visualizer User's Guide for Oracle Event Processing*). In the left pane, navigate to and expand the **Applications** node of the Oracle Event Processing instance to which the application is deployed. Each node under the Applications node is named with the exact application name. ▪ Look at the `MANIFEST.MF` file of the application; the application name is specified by the `Bundle-SymbolicName` header.
`-stage stage`	Specifies the name of the particular stage, attached to the Oracle Event Processing application specified with the `-application` argument, whose event record and playback you want to manage.

Example

The following example shows how to stop the playback of events on the `helloworldAdapter` stage of the `helloworld` application deployed to the specified Oracle Event Processing instance; it is assumed that the playback of events was previously started for the stage:

```
prompt> java wlevs.Admin
    -url service:jmx:msarmi://localhost:9002/jndi/jmxconnector
    -username wlevs -password wlevs
    STOPPLAYBACK -application helloworld -stage helloworldAdapter
```

CONFIGUREPLAYBACK

Configures the parameters associated with the playback of events for a stage of an Oracle Event Processing application.

Syntax

```
java wlevs.Admin
    [ Connection Arguments ]
    [ User Credentials Arguments ]
    [ Common Arguments ]
    CONFIGUREPLAYBACK -application application -stage stage
      [-datasetName datasetname]
      [-storeProvider storeprovidername]
      [-eventTypes eventtypes]
      [-scheduleStartTime sstart] [-scheduleEndTime send | -scheduleDuration sduration]
      [-filterStartTime fstart] [-filterEndTime fend | -filterDuration fduration]
      [-speed speed]
      [-repeat true | false]
```

Table A–38 CONFIGUREPLAYBACK Arguments

Argument	Definition
`-application` *application*	Specifies the name of the Oracle Event Processing application whose event record and playback you want to manage. To get the exact name of your application, you can: ▪ Use `wlevs.Admin` to query for the name (see Section , "Querying for Application and Processor Names"). ▪ Use the Oracle Event Processing Visualizer: Start the Oracle Event Processing Visualizer (see "Starting the Oracle Event Processing Visualizer" in the *Oracle Fusion Middleware Visualizer User's Guide for Oracle Event Processing*). In the left pane, navigate to and expand the **Applications** node of the Oracle Event Processing instance to which the application is deployed. Each node under the Applications node is named with the exact application name. ▪ Look at the `MANIFEST.MF` file of the application; the application name is specified by the `Bundle-SymbolicName` header.
`-stage stage`	Specifies the name of the particular stage, attached to the Oracle Event Processing application specified with the `-application` argument, whose event record and playback you want to manage.
`-datasetName` *datasetname*	Specifies the name of the dataset in which events are recorded.

Table A–38 (Cont.) CONFIGUREPLAYBACK Arguments

Argument	Definition
`-storeProvider` *`storeprovidername`*	Specifies a valid `data-source` name defined in the Oracle Event Processing server `config.xml` file. To select the default BDB provider, leave this argument empty or specify an argument value of `default-provider`. For more information, see "Configuring an Event Store for Oracle Event Processing Server" in the *Oracle Fusion Middleware Developer's Guide for Oracle Event Processing for Eclipse*.
`-eventTypes` *`eventtypes`*	Specifies the comma-separated list of valid event type names for playing back. Event types must be defined in the event type repository.

`-scheduleStartTime` *`sstart`*	Specifies the time when play back should start.
	Express the start time as an XMLSchema `dateTime` value of the form:
	`yyyy-mm-ddThh:mm:ss`
	For example, to specify that play back should start on January 20, 2010, at 5:00am, use the following value:
	`2010-01-20T05:00:00`
	For complete details of the XMLSchema `dateTime` format, see `http://www.w3.org/TR/xmlschema-2/#dateTime-lexical-representation`.

`-scheduleEndTime` *`send`*	Specifies the actual time when the play back should end.
	Express the end time as an XMLSchema `dateTime` value of the form:
	`yyyy-mm-ddThh:mm:ss`
	For example, to specify that play back should end on January 20, 2010, at 6:00pm, use the following value:
	`2010-01-20T18:00:00`
	For complete details of the XMLSchema `dateTime` format, see `http://www.w3.org/TR/xmlschema-2/#dateTime-lexical-representation`.
	Specify `null` if you want the recording to run forever.
	You can specify either `-scheduleEndTime` or `-scheduleDuration`, but not both.

`-scheduleDuration` *`sduration`*	Specifies the duration of time after which event playback for this stage ends. Specify `null` if you want the event playback to run forever. The format is `HH:mm:ss`, such as `01:00:00`. You can specify either `-scheduleEndTime` or `-scheduleDuration`, but not both.

Table A–38 (Cont.) CONFIGUREPLAYBACK Arguments

Argument	Definition

`-filterStartTime` *fstart*	Specifies that only events with record-time greater than or equal to this value will be selected for playback.
	Express the start time as an XMLSchema `dateTime` value of the form:
	`yyyy-mm-ddThh:mm:ss`
	For example, to play back only events with record-time greater than or equal to January 20, 2010, at 5:00am, use the following value:
	`2010-01-20T05:00:00`
	For complete details of the XMLSchema `dateTime` format, see http://www.w3.org/TR/xmlschema-2/#dateTime-lexical-representation.
	Specify `null` if you want to select all events for playback.

`-filterEndTime` *fend*	Specifies only events with record-time less than or equal to this value will be selected for playback.
	Express the end time as an XMLSchema `dateTime` value of the form:
	`yyyy-mm-ddThh:mm:ss`
	For example, to play back only events with record-time less than or equal to January 20, 2010, at 6:00pm, use the following value:
	`2010-01-20T18:00:00`
	For complete details of the XMLSchema `dateTime` format, see http://www.w3.org/TR/xmlschema-2/#dateTime-lexical-representation.
	You can specify either `-filterEndTime` or `-filterDuration`, but not both.

`-filterDuration` *fduration*	Specifies the filter applied to events in the event store. Only events that were recorded during the filter time will be selected for play back. Specify `null` if you want to select all events for playback. The format is `HH:mm:ss`, such as `01:00:00`. You can specify either `-filterEndTime` or `-filterDuration`, but not both.
`-speed` *speed*	Specifies the playback speed as a positive float. The default value is 1, which corresponds to normal speed. A value of 2 means that events will be played back 2 times faster than the original record speed. Similarly, a value of 0.5 means that events will be played back 2 times slower than the original record speed.
`-repeat` *repeat*	Specifies whether to playback events again after the playback of the specified time interval is over. Valid values are `true` or `false`. Default value is `false`. A value of `true` means that the repeat of playback continues an infinite number of times until it is deliberately stopped (see STOPPLAYBACK). A value of `false` means that events will be played back only once.

Example

The examples in this section show how to configure the playback of events of the `helloworldAdapter` of the `helloworld` application deployed to the specified Oracle Event Processing instance.

The following example specifies a start and end time for playback and that the speed of playback should be twice the normal speed and that once the playback of events for the time interval is over, the playback should start again:

```
prompt> java wlevs.Admin
        -url service:jmx:msarmi://localhost:9002/jndi/jmxconnector
        -username wlevs -password wlevs
        CONFIGUREPLAYBACK -application helloworld -stage helloworldAdapter
        -scheduleStartTime 2010-01-20T05:00:00 -scheduleEndTime 2010-01-20T18:00:00
        -speed 2 -repeat true
```

The following example specifies a start and a duration for playback, that the speed of playback is 2 times slower than normal, and that the playback of events should occur only once:

```
prompt> java wlevs.Admin
        -url service:jmx:msarmi://localhost:9002/jndi/jmxconnector
        -username wlevs -password wlevs
        CONFIGUREPLAYBACK -application helloworld -stage helloworldAdapter
        -scheduleStartTime 2010-01-20T05:00:00 -scheduleEndTime 2010-01-20T18:00:00
        -speed 0.5 -repeat false
```

The following example specifies a start and a duration of `null`, which means playback will run forever at normal speed:

```
prompt> java wlevs.Admin
        -url service:jmx:msarmi://localhost:9002/jndi/jmxconnector
        -username wlevs -password wlevs
        CONFIGUREPLAYBACK -application helloworld -stage helloworldAdapter
        -scheduleStartTime 2010-01-20T05:00:00 -scheduleDuration null
```

SCHEDULEPLAYBACK

Configures the schedule parameters associated with playing back of events for a stage of an Oracle Event Processing application.

Syntax

```
java wlevs.Admin
    [ Connection Arguments ]
    [ User Credentials Arguments ]
    [ Common Arguments ]
    SCHEDULEPLAYBACK -application application -stage stage
        [-scheduleStartTime startime]
        [-scheduleEndTime endtime | -scheduleDuration duration]
```

Table A–39 SCHEDULEPLAYBACK Arguments

Argument	Definition
`-application` *application*	Specifies the name of the Oracle Event Processing application whose event record and playback you want to manage. To get the exact name of your application, you can: ▪ Use `wlevs.Admin` to query for the name (see Section , "Querying for Application and Processor Names"). ▪ Use the Oracle Event Processing Visualizer: Start the Oracle Event Processing Visualizer (see "Starting the Oracle Event Processing Visualizer" in the *Oracle Fusion Middleware Visualizer User's Guide for Oracle Event Processing*). In the left pane, navigate to and expand the **Applications** node of the Oracle Event Processing instance to which the application is deployed. Each node under the Applications node is named with the exact application name. ▪ Look at the `MANIFEST.MF` file of the application; the application name is specified by the `Bundle-SymbolicName` header.
`-stage` *stage*	Specifies the name of the particular stage, attached to the Oracle Event Processing application specified with the `-application` argument, whose event record and playback you want to manage.

`-scheduleStartTime` *starttime*	Specifies the time when play back should start.
	Express the start time as an XMLSchema `dateTime` value of the form:
	`yyyy-mm-ddThh:mm:ss`
	For example, to specify that play back should start on January 20, 2010, at 5:00am, use the following value:
	`2010-01-20T05:00:00`
	For complete details of the XMLSchema `dateTime` format, see http://www.w3.org/TR/xmlschema-2/#dateTime-lexical-representation.

`-scheduleEndTime` *endtime*	Specifies the actual time when the play back should end.
	Express the end time as an XMLSchema `dateTime` value of the form:
	`yyyy-mm-ddThh:mm:ss`
	For example, to specify that play back should end on January 20, 2010, at 6:00pm, use the following value:
	`2010-01-20T18:00:00`
	For complete details of the XMLSchema `dateTime` format, see http://www.w3.org/TR/xmlschema-2/#dateTime-lexical-representation.
	Specify `null` if you want the recording to run forever.
	You can specify either `-scheduleEndTime` or `-scheduleDuration`, but not both.

`-scheduleDuration` *duration*	Specifies the duration of time after which event playback for this stage ends. Specify `null` if you want the recording to run forever. The format is `HH:mm:ss`, such as `01:00:00`. You can specify either `-scheduleEndTime` or `-scheduleDuration`, but not both.

Example

The examples in this section show how to configure the schedule of playback of events of the `helloworldAdapter` of the `helloworld` application deployed to the specified Oracle Event Processing instance.

The following example specifies a start and end time for event playback:

```
prompt> java wlevs.Admin
```

```
    -url service:jmx:msarmi://localhost:9002/jndi/jmxconnector
    -username wlevs -password wlevs
    SCHEDULEPLAYBACK -application helloworld -stage helloworldAdapter
    -scheduleStartTime 2010-01-20T05:00:00 -scheduleEndndTime 2010-01-20T18:00:00
```

LISTPLAYBACK

Lists the event playback configuration for any particular stage of an Oracle Event Processing application.

Syntax

```
java wlevs.Admin
    [ Connection Arguments ]
    [ User Credentials Arguments ]
    [ Common Arguments ]
    LISTPLAYBACK -application application -stage stage
```

Table A–40 LISTPLAYBACK Arguments

Argument	Definition
`-application` *application*	Specifies the name of the Oracle Event Processing application whose event record and playback you want to manage. To get the exact name of your application, you can: ▪ Use `wlevs.Admin` to query for the name (see Section , "Querying for Application and Processor Names"). ▪ Use the Oracle Event Processing Visualizer: Start the Oracle Event Processing Visualizer (see "Starting the Oracle Event Processing Visualizer" in the *Oracle Fusion Middleware Visualizer User's Guide for Oracle Event Processing*). In the left pane, navigate to and expand the **Applications** node of the Oracle Event Processing instance to which the application is deployed. Each node under the Applications node is named with the exact application name. ▪ Look at the `MANIFEST.MF` file of the application; the application name is specified by the `Bundle-SymbolicName` header.
`-stage` *stage*	Specifies the name of the particular stage, attached to the Oracle Event Processing application specified with the `-application` argument, whose event record and playback you want to manage.

Example

The following example shows how to list the event playback configuration on the `helloworldAdapter` stage of the `helloworld` application deployed to the specified Oracle Event Processing instance:

```
prompt> java wlevs.Admin
    -url service:jmx:msarmi://localhost:9002/jndi/jmxconnector
    -username wlevs -password wlevs
    LISTPLAYBACK -application helloworld -stage helloworldAdapter
```

Commands for Monitoring Throughput and Latency

Table A–41 is an overview of commands for monitoring throughput and latency in an Oracle Event Processing application. Subsequent sections describe command syntax and arguments, and provide an example for each command.

Table A–41 Overview of Commands for Monitoring Throughput and Latency

Command	Description
MONITORAVGLATENCY	Monitors the *average amount of time it takes an event to pass through specified path of the EPN, or latency*.
MONITORMAXLATENCY	Monitors the *maximum amount of time it takes an event to pass through specified path of the EPN, or latency*.
MONITORAVGLATENCYTHRESHOLD	Monitors whether the average latency of events flowing through a path of the EPN crosses a specified threshold.
MONITORAVGTHROUGHPUT	Monitors the number of events flowing through the entry or exit points of a specified stage.

MONITORAVGLATENCY

Monitors the average amount of time, or *latency*, it takes an event to pass through a specified path of the EPN of the specified application.

You specify the start and end stages of the path, and whether it should start or end at the entry or exit points of each respective stage. If you specify the same stage for the start and end of the path, you can monitor the latency of events flowing through a single stage.

Syntax

```
java wlevs.Admin
     [ Connection Arguments ]
     [ User Credentials Arguments ]
     [ Common Arguments ]
     MONITORAVGLATENCY -application application
     -startStage startStage -startStagePoint stagePoint
     -endStage endStage -endStagePoint stagePoint
     -avgInterval avgInterval -timeUnit timeUnit
```

Table A–42 MONITORAVGLATENCY Arguments

Argument	Definition
`-application application`	Specifies the name of the Oracle Event Processing application whose throughput and latency you want to monitor. To get the exact name of your application, you can: ▪ Use `wlevs.Admin` to query for the name (see Section , "Querying for Application and Processor Names"). ▪ Use the Oracle Event Processing Visualizer: Start the Oracle Event Processing Visualizer (see "Starting the Oracle Event Processing Visualizer" in the *Oracle Fusion Middleware Visualizer User's Guide for Oracle Event Processing*). In the left pane, navigate to and expand the **Applications** node of the Oracle Event Processing instance to which the application is deployed. Each node under the Applications node is named with the exact application name. ▪ Look at the `MANIFEST.MF` file of the application; the application name is specified by the `Bundle-SymbolicName` header.
`-startStage startStage`	Specifies the name of the stage that starts the path for which you want to monitor latency. The stage is in the application specified by the `-application` option.
`-startStagePoint startStagePoint`	Specifies the specific starting point for monitoring latency of the specified start stage. You can start monitoring from the entry or exit point of the start stage. Valid values are `entry` and `exit`. Default value is `entry`.

Table A–42 (Cont.) MONITORAVGLATENCY Arguments

Argument	Definition
`-endStage endStage`	Specifies the name of the stage that ends the path for which you want to monitor latency. The stage is in the application specified by the `-application` option.
`-endStagePoint endStagePoint`	Specifies the specific ending point for monitoring latency of the specified end stage. You can end monitoring from the entry or exit point of the end stage. Valid values are `entry` and `exit`. Default value is `entry`.
`-avgInterval` *avgInterval*	Specifies the average interval across which average latency is calculated. Specify the units with the `-timeUnit` option; default is milliseconds. Default value is 100.
`-timeUnit timeUnit`	Specifies the time unit for the latency calculation. Valid values are `MILLISECONDS` and `SECONDS`. Default value is `MILLISECONDS`.

Example

The following example shows how to monitor the average latency of events flowing through the `eplprocessor` component, from entry point to exit point, of the `helloworld` application. Note that because the same stage is specified for both the start and end stages (`eplprocessor`), the latency monitoring is happening for just the events flowing through a single stage:

```
prompt> java wlevs.Admin
        -url service:jmx:msarmi://localhost:9002/jndi/jmxconnector
        -username wlevs -password wlevs
        MONITORAVGLATENCY -application helloworld -startStage eplprocessor
        -startStagePoint entry -endStage eplprocessor -endStagePoint exit
        -avgInterval 100 -timeUnit MILLISECONDS
```

MONITORAVGLATENCYTHRESHOLD

Specifies whether the average latency of events between the start- and end-points of a path crosses a specified threshold.

You specify the start and end stages of the path, and whether it should start or end at the entry or exit points of each respective stage. If you specify the same stage for the start and end of the path, you can monitor the latency threshold of events flowing through a single stage.

Syntax

```
java wlevs.Admin
    [ Connection Arguments ]
    [ User Credentials Arguments ]
    [ Common Arguments ]
    MONITORAVGLATENCYTHRESHOLD -application application
    -startStage startStage -startStagePoint stagePoint
    -endStage endStage -endStagePoint stagePoint
    -avgInterval avgInterval -timeUnit timeUnit -threshold threshold
```

Table A–43 MONITORAVGLATENCYTHRESHOLD Arguments

Argument	Definition
`-application` *application*	Specifies the name of the Oracle Event Processing application whose throughput and latency you want to monitor. To get the exact name of your application, you can: ■ Use `wlevs.Admin` to query for the name (see Section , "Querying for Application and Processor Names"). ■ Use the Oracle Event Processing Visualizer: Start the Oracle Event Processing Visualizer (see "Starting the Oracle Event Processing Visualizer" in the *Oracle Fusion Middleware Visualizer User's Guide for Oracle Event Processing*). In the left pane, navigate to and expand the **Applications** node of the Oracle Event Processing instance to which the application is deployed. Each node under the Applications node is named with the exact application name. ■ Look at the `MANIFEST.MF` file of the application; the application name is specified by the `Bundle-SymbolicName` header.
`-startStage startStage`	Specifies the name of the stage that starts the path for which you want to monitor the latency threshold. The stage is in the application specified by the `-application` option.
`-startStagePoint startStagePoint`	Specifies the specific starting point for monitoring the latency threshold of the specified start stage. You can start monitoring from the entry or exit point of the start stage. Valid values are `entry` and `exit`. Default value is `entry`.
`-endStage endStage`	Specifies the name of the stage that ends the path for which you want to monitor the latency threshold. The stage is in the application specified by the `-application` option.
`-endStagePoint endStagePoint`	Specifies the specific ending point for monitoring the latency threshold of the specified end stage. You can end monitoring from the entry or exit point of the end stage. Valid values are `entry` and `exit`. Default value is `entry`.
`-avgInterval` *avgInterval*	Specifies the average interval across which average the latency threshold is calculated. Default value is 100. Specify the units with the `-timeUnit` option; default is milliseconds.
`-timeUnit timeUnit`	Specifies the time unit for the latency calculation. Valid values are `MILLISECONDS` and `SECONDS`. Default value is `MILLISECONDS`.
`-threshold` *threshold*	Specifies the threshold value above which the metric event will be outputted at the end of every average interval. Default is 100. Specify the units with the `-timeUnit` option; default is milliseconds.

Example

The following example shows how to monitor the average latency threshold of events above 10 seconds average latency on the `eplprocessor` stage, from entry point to exit point, of the `helloworld` application.

```
prompt> java wlevs.Admin
        -url service:jmx:msarmi://localhost:9002/jndi/jmxconnector
        -username wlevs -password wlevs
        MONITORAVGLATENCY -application helloworld -startStage eplprocessor
        -startStagePoint entry -endStage eplprocessor -endStagePoint exit
        -avgInterval 100 -timeUnit MILLISECONDS -threshold 100
```

MONITORMAXLATENCY

Monitors the maximum latency of events flowing through a specified path of the EPN of the specified application.

You specify the start and end stages of the path, and whether it should start or end at the entry or exit points of each respective stage. If you specify the same stage for the start and end of the path, you can monitor the maximum latency of events flowing through a single stage.

Syntax

```
java wlevs.Admin
    [ Connection Arguments ]
    [ User Credentials Arguments ]
    [ Common Arguments ]
    MONITORMAXLATENCY -application application
    -startStage startStage -startStagePoint stagePoint
    -endStage endStage -endStagePoint stagePoint
    -maxInterval maxInterval -timeUnit timeUnit
```

Table A–44 MONITORMAXLATENCY Arguments

Argument	Definition
`-application` *application*	Specifies the name of the Oracle Event Processing application whose throughput and latency you want to monitor. To get the exact name of your application, you can: ■ Use `wlevs.Admin` to query for the name (see Section , "Querying for Application and Processor Names"). ■ Use the Oracle Event Processing Visualizer: Start the Oracle Event Processing Visualizer (see "Starting the Oracle Event Processing Visualizer" in the *Oracle Fusion Middleware Visualizer User's Guide for Oracle Event Processing*). In the left pane, navigate to and expand the **Applications** node of the Oracle Event Processing instance to which the application is deployed. Each node under the Applications node is named with the exact application name. ■ Look at the `MANIFEST.MF` file of the application; the application name is specified by the `Bundle-SymbolicName` header.
`-startStage startStage`	Specifies the name of the stage that starts the path for which you want to monitor the maximum latency. The stage is in the application specified by the `-application` option.
`-startStagePoint startStagePoint`	Specifies the specific starting point for monitoring the maximum latency of the specified start stage. You can start monitoring from the entry or exit point of the start stage. Valid values are `entry` and `exit`. Default value is `entry`.
`-endStage endStage`	Specifies the name of the stage that ends the path for which you want to monitor the maximum latency. The stage is in the application specified by the `-application` option.
`-endStagePoint endStagePoint`	Specifies the specific ending point for monitoring the maximum latency of the specified end stage. You can end monitoring from the entry or exit point of the end stage. Valid values are `entry` and `exit`. Default value is `entry`.
`-maxInterval` *maxInterval*	Specifies the interval across which maximum latency is calculate. Default value is 100. Specify the units with the `-timeUnit` option; default is milliseconds.
`-timeUnit timeUnit`	Specifies the time unit for the maximum calculation. Valid values are `MILLISECONDS` and `SECONDS`. Default value is `MILLISECONDS`.

Example

The following example shows how to monitor the maximum latency of events flowing through the `eplprocessor` stage, from entry point to exit point, of the `helloworld` application:

```
prompt> java wlevs.Admin
        -url service:jmx:msarmi://localhost:9002/jndi/jmxconnector
        -username wlevs -password wlevs
        MONITORMAXLATENCY -application helloworld -startStage eplprocessor
        -startStagePoint entry -endStage eplprocessor -endStagePoint exit
        -maxInterval 100 -timeUnit MILLISECONDS
```

MONITORAVGTHROUGHPUT

Monitors the average number of events flowing through the entry or exit point of a stage of the EPN of the specified application.

Syntax

```
java wlevs.Admin
    [ Connection Arguments ]
    [ User Credentials Arguments ]
    [ Common Arguments ]
    MONITORAVGTHROUGHPUT -application application
    -stage stage -StagePoint stagePoint
    -throughputInterval throughputInterval -avgInterval avgInterval
    -timeUnit timeUnit
```

Table A–45 MONITORAVGLATENCY Arguments

Argument	Definition
`-application` *application*	Specifies the name of the Oracle Event Processing application whose throughput and latency you want to monitor. To get the exact name of your application, you can: ▪ Use `wlevs.Admin` to query for the name (see Section , "Querying for Application and Processor Names"). ▪ Use the Oracle Event Processing Visualizer: Start the Oracle Event Processing Visualizer (see "Starting the Oracle Event Processing Visualizer" in the *Oracle Fusion Middleware Visualizer User's Guide for Oracle Event Processing*). In the left pane, navigate to and expand the **Applications** node of the Oracle Event Processing instance to which the application is deployed. Each node under the Applications node is named with the exact application name. ▪ Look at the `MANIFEST.MF` file of the application; the application name is specified by the `Bundle-SymbolicName` header.
`-stage stage`	Specifies the name of the stage for which you want to monitor throughput of events. The stage is in the application specified by the `-application` option.
`-stagePoint stagePoint`	Specifies whether you want to monitor throughput at the entry- or exit- point of the specified stage. Valid values are `entry` and `exit`. Default value is `entry`.
`-throughputInterval` *throughputInterval*	Specifies the throughput interval across which throughput is calculated. Default value is 100. Specify the units with the `-timeUnit` option; default is milliseconds.

Table A–45 (Cont.) MONITORAVGLATENCY Arguments

Argument	Definition
`-avgInterval avgInterval`	Specifies the average interval across which average throughput is calculated. Default value is 100. Specify the units with the `-timeUnit` option; default is milliseconds.
`-timeUnit timeUnit`	Specifies the time unit for the throughput calculation. Valid values are `MILLISECONDS` and `SECONDS`. Default value is `MILLISECONDS`.

Example

The following example shows how to monitor the number of events flowing through the entry point of the `eplprocessor` stage of the `helloworld` application:

```
prompt> java wlevs.Admin
        -url service:jmx:msarmi://localhost:9002/jndi/jmxconnector
        -username wlevs -password wlevs
        MONITORMAXLATENCY -application helloworld
        -stage eplprocessor -stagePoint entry
        -throughputInterval 100 -avgInterval 100 -timeUnit MILLISECONDS
```

Commands for Managing Configuration History

Table A–46 is an overview of commands that manage the configuration history of Oracle Event Processing components. For more information, see Section , "Configuration History Management".

Subsequent sections describe command syntax and arguments, and provide an example for each command.

Table A–46 Overview of Commands for Managing Configuration History

Command	Description
CONFIGHISTORY	Returns the list of configuration history management commands.
DELETECONFIGCHANGEHISTORY	Removes change records for a specified time period.
LISTCHANGERECORDS	Returns a list of the change records of an application.
LISTRESOURCEREVISIONS	Returns a list of the configuration resource revisions of an application.
UNDOCONFIGCHANGE	Rolls back a change record specified by change record ID.

CONFIGHISTORY

Returns the list of configuration history management commands.

Syntax

```
java wlevs.Admin
    [ Connection Arguments ]
    [ User Credentials Arguments ]
    [ Common Arguments ]
    CONFIGHISTORY
```

Example

The following example shows how to list the configuration history management commands:

```
prompt> java wlevs.Admin
        -url service:jmx:msarmi://localhost:9002/jndi/jmxconnector
        -username wlevs -password wlevs
        CONFIGHISTORY
```

DELETECONFIGCHANGEHISTORY

Returns the list of configuration history management commands.

Syntax

```
java wlevs.Admin
    [ Connection Arguments ]
    [ User Credentials Arguments ]
    [ Common Arguments ]
    DELETECONFIGCHANGEHISTORY -application application -startTime starttime -endTime endtime
```

Table A–47 DELETECONFIGCHANGEHISTORY Arguments

Argument	Definition
`-application` *application*	Specifies the name of the Oracle Event Processing application whose Oracle CQL rules you want to manage. To get the exact name of your application, you can: ▪ Use `wlevs.Admin` to query for the name (see Section , "Querying for Application and Processor Names"). ▪ Use the Oracle Event Processing Visualizer: Start the Oracle Event Processing Visualizer (see "Starting the Oracle Event Processing Visualizer" in the *Oracle Fusion Middleware Visualizer User's Guide for Oracle Event Processing*). In the left pane, navigate to and expand the **Applications** node of the Oracle Event Processing instance to which the application is deployed. Each node under the Applications node is named with the exact application name. ▪ Look at the `MANIFEST.MF` file of the application; the application name is specified by the `Bundle-SymbolicName` header.
`-startTime` *starttime*	Specifies the beginning of the time period to delete change records. The format is `MM-dd-yyyy:HH:mm:ss`, such as `10-20-2007:11:22:07`.
`-endTime` *end-ime*	Specifies the end of the time period to delete change records. The format is `MM-dd-yyyy:HH:mm:ss`, such as `10-20-2007:11:22:07`.

Example

The following example shows how to list the configuration history management commands:

```
prompt> java wlevs.Admin
        -url service:jmx:msarmi://localhost:9002/jndi/jmxconnector
        -username wlevs -password wlevs
        CONFIGHISTORY
```

LISTCHANGERECORDS

Returns a list of the change records of an application.

Syntax

```
java wlevs.Admin
    [ Connection Arguments ]
    [ User Credentials Arguments ]
    [ Common Arguments ]
```

```
LISTCHANGERECORDS -application application -startTime starttime -endTime endtime
```

Table A–48 GETRULE Arguments

Argument	Definition
`-application` *application*	Specifies the name of the Oracle Event Processing application whose change records you want to browse. To get the exact name of your application, you can: ▪ Use `wlevs.Admin` to query for the name (see Section , "Querying for Application and Processor Names"). ▪ Use the Oracle Event Processing Visualizer: Start the Oracle Event Processing Visualizer (see "Starting the Oracle Event Processing Visualizer" in the *Oracle Fusion Middleware Visualizer User's Guide for Oracle Event Processing*). In the left pane, navigate to and expand the **Applications** node of the Oracle Event Processing instance to which the application is deployed. Each node under the Applications node is named with the exact application name. ▪ Look at the `MANIFEST.MF` file of the application; the application name is specified by the `Bundle-SymbolicName` header.
`-startTime` *starttime*	Specifies the beginning of the time period to filter the display of change records. The format is `MM-dd-yyyy:HH:mm:ss`, such as `10-20-2007:11:22:07`.
`-endTime` *end-ime*	Specifies the end of the time period to filter the display of change records. The format is `MM-dd-yyyy:HH:mm:ss`, such as `10-20-2007:11:22:07`.

Example

The following example shows how to list all the change records created between 11:10:07 and 11:22:07 on 20 November 2007 for the application `helloworld`:

```
prompt> java wlevs.Admin
        -url service:jmx:msarmi://localhost:9002/jndi/jmxconnector
        -username wlevs -password wlevs
        LISTCHANGERECORDS -application helloworld startTime 10-20-2007:11:10:07
        -endTime 10-20-2007:11:22:07
```

LISTRESOURCEREVISIONS

Returns a list of the configuration resource revisions of an application.

Syntax

```
java wlevs.Admin
    [ Connection Arguments ]
    [ User Credentials Arguments ]
    [ Common Arguments ]
    LISTRESOURCEREVISIONS -application application -startTime starttime -endTime endtime
```

Table A–49 GETRULE Arguments

Argument	Definition
`-application` *application*	Specifies the name of the Oracle Event Processing application whose resource revisions you want to browse. To get the exact name of your application, you can: ■ Use `wlevs.Admin` to query for the name (see Section , "Querying for Application and Processor Names"). ■ Use the Oracle Event Processing Visualizer: Start the Oracle Event Processing Visualizer (see "Starting the Oracle Event Processing Visualizer" in the *Oracle Fusion Middleware Visualizer User's Guide for Oracle Event Processing*). In the left pane, navigate to and expand the **Applications** node of the Oracle Event Processing instance to which the application is deployed. Each node under the Applications node is named with the exact application name. ■ Look at the `MANIFEST.MF` file of the application; the application name is specified by the `Bundle-SymbolicName` header.
`-startTime` *starttime*	Specifies the beginning of the time period to filter the list of resource revisions. The format is `MM-dd-yyyy:HH:mm:ss`, such as `10-20-2007:11:22:07`.
`-endTime` *end-ime*	Specifies the end of the time period to filter the list of resource revisions. The format is `MM-dd-yyyy:HH:mm:ss`, such as `10-20-2007:11:22:07`.

Example

The following example shows how to list all the resource revisions created between 11:10:07 and 11:22:07 on 20 November 2007 for the application `helloworld`:

```
prompt> java wlevs.Admin
        -url service:jmx:msarmi://localhost:9002/jndi/jmxconnector
        -username wlevs -password wlevs
        LISTRESOURCEREVISIONS -application helloworld startTime 10-20-2007:11:10:07
        -endTime 10-20-2007:11:22:07
```

UNDOCONFIGCHANGE

Rolls back the changes defined by the change record specified by change record ID.

Syntax

```
java wlevs.Admin
    [ Connection Arguments ]
    [ User Credentials Arguments ]
    [ Common Arguments ]
    UNDOCONFIGCHANGE -application application -recordId changerecordid
```

Table A–50 GETRULE Arguments

Argument	Definition
`-application` *application*	Specifies the name of the Oracle Event Processing application whose change records you want to undo. To get the exact name of your application, you can: ■ Use `wlevs.Admin` to query for the name (see Section , "Querying for Application and Processor Names"). ■ Use the Oracle Event Processing Visualizer: Start the Oracle Event Processing Visualizer (see "Starting the Oracle Event Processing Visualizer" in the *Oracle Fusion Middleware Visualizer User's Guide for Oracle Event Processing*). In the left pane, navigate to and expand the **Applications** node of the Oracle Event Processing instance to which the application is deployed. Each node under the Applications node is named with the exact application name. ■ Look at the `MANIFEST.MF` file of the application; the application name is specified by the `Bundle-SymbolicName` header.
`-recordId` *changerecordid*	Specifies the identifier of the change record to undo. To get the change record identifier, you can use: ■ "LISTCHANGERECORDS" on page A-56 ■ "LISTRESOURCEREVISIONS" on page A-57

Example

The following example shows how to roll back all the resource revisions created between 11:10:07 and 11:22:07 on 20 November 2007 for the application `helloworld`:

```
prompt> java wlevs.Admin
        -url service:jmx:msarmi://localhost:9002/jndi/jmxconnector
        -username wlevs -password wlevs
        UNDOCONFIGCHANGE -application helloworld -recordId tr.1267607521409.10110
```

B

Deployer Command-Line Reference

This appendix provides a reference to the Oracle Event Processing `Deployer` command-line utility, which you can use to deploy Oracle Event Processing applications.

This appendix includes the following sections:

Overview of Using the Deployer Command-Line Utility

The `Deployer` is a Java-based deployment utility that provides administrators and developers command-line based operations for deploying Oracle Event Processing applications. In the context of Oracle Event Processing deployment, an application is defined as an OSGi bundle at `http://www.osgi.org/` JAR file that contains the following artifacts:

- The compiled Java class files that implement some of the components of the application, such as the adapters, adapter factory, and POJO that contains the business logic.
- One or more Oracle Event Processing configuration XML files that configure the components of the application, such as the processor, adapter, or streams.

  The configuration files must be located in the `META-INF/wlevs` directory of the OSGi bundle JAR file.
- An EPN assembly file that describes all the components of the application and how they are connected to each other. The EPN assembly file extends the standard Spring context file.

  The EPN assembly file must be located in the `META-INF/spring` directory of the OSGi bundle JAR file.
- A `MANIFEST.MF` file that describes the contents of the JAR.

See "Assembling an Oracle Event Processing Application" in the *Oracle Fusion Middleware Developer's Guide for Oracle Event Processing for Eclipse* for detailed instructions on creating this deployment bundle.

The Deployer utility uses HTTP to connect to Oracle Event Processing, which means that you must configure Jetty for the server instance to which you are deploying your application. For details, see "Configuring Jetty" in the *Oracle Fusion Middleware Administrator's Guide for Oracle Event Processing*.

Oracle Event Processing uses the `deployments.xml` file to internally maintain its list of deployed application OSGi bundles. This file is located in the *`DOMAIN_DIR/servername`* directory, where *`DOMAIN_DIR`* refers to the main domain directory corresponding to the server instance to which you are deploying your application and *`servername`* refers to the server instance itself. For more information, see "Oracle Event Processing Schemas" in the *Oracle Fusion Middleware Developer's Guide for Oracle Event Processing for Eclipse*.

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

Configuring the Deployer Utility Environment

Before you can use the Deployer utility, you must configure your environment appropriately.

To configure the Deployer utility environment:

1. Install and configure the Oracle Event Processing software, as described in "Installing Oracle Event Processing" in the *Oracle Fusion Middleware Getting Started Guide for Oracle Event Processing*.
2. Open a command window and set your environment as described in "Setting Your Development Environment" in the *Oracle Fusion Middleware Developer's Guide for Oracle Event Processing for Eclipse*.
3. Update your `CLASSPATH` variable to include the `wlevsdeploy.jar JAR` file, located in the *`ORACLE_CEP_HOME`*`/ocep_11.1/bin` directory where, *`ORACLE_CEP_HOME`* refers to the main Oracle Event Processing installation directory, such as `/oracle_cep`.

Running the Deployer Utility Remotely

Sometimes it is useful to run the `Deployer` utility on a computer different from the computer on which Oracle Event Processing is installed and running.

To run the Deployer utility remotely:

1. Copy the following JAR files from the computer on which Oracle Event Processing is installed to the computer on which you want to run the deployer utility; you can copy the JAR files to the directory name of your choice:
   - *`ORACLE_CEP_HOME`*`/ocep_11.1/bin/`wlevsdeploy.jar

   where *`ORACLE_CEP_HOME`* refers to the main directory into which you installed Oracle Event Processing.
2. Set your `CLASSPATH` in one of the following ways:
   - Implicitly set your `CLASSPATH` by using the `-jar` argument when you run the utility; set the argument to the *`NEW_DIRECTORY`*`/wlevsdeploy.jar` file, where *`NEW_DIRECTORY`* refers to the directory on the remote computer into which you copied the required JAR file. When you use the `-jar` argument, you do not specify the `Deployer` utility name at the command line.

- Explicitly update your `CLASSPATH` by adding the JAR file you copied to the remote computer to your `CLASSPATH` environment variable:

3. Invoke the `Deployer` utility as described in the next section.

Syntax for Invoking the Deployer Utility

The syntax for using the Deployer utility is as follows:

```
java -jar wlevsdeploy.jar
    [Connection Arguments]
    [User Credential Arguments]
    [Deployment Commands]
```

The following sections describe the various arguments and commands you can use with the Deployer utility. See Appendix , "Examples of Using the Deployer Utility" for specific examples of using the utility.

Connection Arguments

Table B–1 lists the connection arguments you can specify with the Deployer utility.

Table B–1 Connection Arguments

Argument	Description

`-url` *`url`*	Specifies the URL of the deployer of the Oracle Event Processing instance to which you want to deploy the OSGI bundle.
	The URL takes the following form:
	`http://`*`host`*`:`*`port`*`/wlevsdeployer`
	
	where:
	▪ *`host`* refers to the hostname of the computer on which Oracle Event Processing is running.
	▪ *`port`* refers to the port number to which Oracle Event Processing listens. Its value is `9002` by default. This port is specified in the `config.xml` file that describes your Oracle Event Processing domain, located in the *`DOMAIN_DIR`*`/config` directory, where *`DOMAIN_DIR`* refers to your domain directory.
	The port number is the value of the `<Port>` child element of the `<Netio>` element:
	`<Netio>`
	`<Name>NetIO</Name>`
	`<Port>9002</Port>`
	`</Netio>`
	
	If you configure the Oracle Event Processing server for SSL-only connections (Section , "Configuring HTTPS-Only Connections for Oracle Event Processing Server"), use the value of the `<Port>` child element of the SSL `<Netio>` element:
	`<Netio>`
	`<name>sslNetIo</name>`
	`<port>9003</port>`
	`<ssl-config-bean-name>sslConfig</ssl-config-bean-name>`
	`</Netio>`
	
	For example, if Oracle Event Processing is running on host `ariel` at port `9002`, then the URL would be:
	`http://ariel:9002/wlevsdeployer`

User Credential Arguments

Table B–2 lists the user credential arguments you can specify with the Deployer utility.

Table B–2 User Credential Arguments

Argument	Description
`-user` *username*	Username of the Oracle Event Processing administrator. If you supply the `-user` option but you do not supply a corresponding `-password` option, the Deployer utility prompts you for the password.
`-password` *password*	Password of the Oracle Event Processing administrator. **NOTE:** this argument is deprecated and may be removed in a later release. Oracle recommends that you do not use this argument.

Deployment Commands

Table B–3 lists the deployment commands you can specify with the Deployer utility.

Table B–3 Deployment Commands

Command	Description
`-encrypt`	Encrypts the username and password and writes it to output file.
`-encryptoutfile` *encryptoutfile*	Specifies that *encryptoutfile* should be used to write encrypted the user and password.

`-group` *groupname*	Specifies that the deploy command (install, uninstall, update, suspend, or resume) applies to a target group, or more specifically, to the set of running servers within that group.
	To specify the domain group, use the keyword `all`, such as:
	`-group all`
	To specify a custom group, simply specify the name of the group:
	`-group my_group`
	Note: You may only deploy to a group if the server is part of a multi-server domain (that is, if clustering is enabled). You may not deploy to a group if the server is part of a standalone-server domain (that is, if clustering is disabled). For more information, see Section , "Overview of Oracle Event Processing Multi-Server Domain Administration".

`-help`	Prints a message describe command syntax and arguments.
`-install` *bundle*	Installs the specified OSGi bundle to the specified Oracle Event Processing instance. The *bundle* parameter refers to a filename that is local to the computer from which you execute the `Deployer` utility. Be sure to specify the full pathname of the bundle if it is not located relative to the directory from which you are running the Deployer utility. In particular, Oracle Event Processing: ▪ Copies the specified bundle to the domain directory. ▪ Searches the `META-INF/wlevs` directory in the bundle for the component configuration files and extracts them to the domain directory. ▪ Updates the internal deployment registry. ▪ Starts the application. The incoming adapters immediately start receiving data.
`-resume` *name*	Resumes a previously suspended OSGI bundle on the specified Oracle Event Processing instance; the configured adapters once again start immediately receiving incoming data. The *name* parameter refers to the symbolic name of the OSGi bundle that you want to stop. The symbolic name is the value of the `Bundle-SymbolicName` header in the bundle's `MANIFEST.MF` file: `Bundle-SymbolicName: myApp`

Table B–3 (Cont.) Deployment Commands

Command	Description
`-status` *name*	Returns status information about a currently installed OSGi bundle. The *name* parameter refers to the symbolic name of the OSGi bundle for which you want status information. The symbolic name is the value of the `Bundle-SymbolicName` header in the bundle's `MANIFEST.MF` file: `Bundle-SymbolicName: myApp`
`-suspend` *name*	Suspends a currently running OSGI bundle which was previously installed to the specified Oracle Event Processing instance. The *name* parameter refers to the symbolic name of the OSGi bundle that you want to start. The symbolic name is the value of the `Bundle-SymbolicName` header in the bundle's `MANIFEST.MF` file: `Bundle-SymbolicName: myApp`
`-uninstall` *name*	Removes the existing bundle from the specified Oracle Event Processing instance. The *name* parameter refers to the symbolic name of the OSGi bundle that you want to remove. The symbolic name is the value of the `Bundle-SymbolicName` header in the bundle's `MANIFEST.MF` file: `Bundle-SymbolicName: myApp` In particular, Oracle Event Processing: ■ Removes the specified OSGi bundle from the domain directory. ■ Removes the bundles from the internal deployment registry.
`-update` *bundle*	Updates the existing OSGi bundle with new application code. The *bundle* parameter refers to a filename that is local to the computer from which you execute the `Deployer` utility. Be sure to specify the full pathname of the bundle if it is not located relative to the directory from which you are running the Deployer utility. In particular, Oracle Event Processing: ■ Copies the updated bundles to the domain directory. ■ Searches the `META-INF/wlevs` directory in the updated bundle for the updated component configuration files and extracts them to the domain directory. ■ Updates the internal deployment registry with the updated information.
`-userconfigfile` *userconfigfile*	Specifies that *userconfigfile* (security-config.xml) should be used to retrieve encrypted username and password from the file.
`-userkeyfile` *userkeyfile*	Specifies that *userkeyfile* (.msainternal.dat) should be used to get the encryption key used to encrypt the password in the user config file.

Examples of Using the Deployer Utility

The following examples show how to use the Deployer utility. In all the examples, Oracle Event Processing is running on host `ariel`, listening at port `9002`, and the username/password of the server administrator is `wlevs/wlevs`, respectively. For clarity, the examples are shown on multiple lines; however, when you run the command, enter all arguments and commands on a single line.

```
prompt> java -jar wlevsdeploy.jar
        -url http://ariel:9002/wlevsdeployer -user wlevs -password wlevs
        -install /application/bundles/com.my.exampleApp_1.0.0.0.jar
```

The preceding example shows how to install an OSGi bundle called `com.my.exampleApp_1.0.0.0.jar`, located in the `/application/bundles` directory.

The next command shows how to resume this application after it has been suspended:

```
prompt> java com.bea.wlevs.deployment.Deployer
```

```
        -url http://ariel:9002/wlevsdeployer -user wlevs -password wlevs
        -resume exampleApp
```

The next example shows how to uninstall the application, which removes all traces of it from the domain directory:

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

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

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

C

Security Utilities Command-Line Reference

This appendix provides a reference to the Oracle Event Processing security utilities, including cssconfig, encryptMSAConfig, and GrabCert -- utilities for generating security configuration files, encrypting cleartext passwords, and generating a trust keystore.

This appendix includes the following sections:

The cssconfig Command-Line Utility

Use the `cssconfig` command-line utility to generate a security configuration file (`security.xml`) that uses a password policy.

The `cssconfig` utility is located in the *`ORACLE_CEP_HOME`*`/ocep_11.1/bin` directory, where *`ORACLE_CEP_HOME`* is the main Oracle Event Processing installation directory, such as `d:\oracle_cep`. The utility comes in two flavors:

- `cssconfig.cmd` (Windows)
- `cssconfig.sh` (UNIX)

The Unix version of this utility starts with the `#!/bin/ksh` directive. On most Unix systems, this forces the Korn Shell program to be used when using the utility. If the `ksh` program is not present in the `bin` directory or if the shell language used cannot properly execute the utility, run the utility as shown below:

```
prompt> $PATH_TO_KSH_BIN/ksh -c cssconfig.sh
```

where *`PATH_TO_KSH_BIN`* is the fully qualified path to the `ksh` program.

cssconfig Syntax

```
cssconfig -p propertyfile [-c configfile] -i inputkeyfile [-d]
```

where:

- *`propertyfile`* is a file that contains security configuration properties provided by the user to define the required configuration. This option is required. See Example 10–1 for an example.

- *configfile* is the name of the generated file. This property is optional; default value is security.xml.
- *inputkeyfile* is the fully qualified name of the input key file used to generate the security configuration file. Set this option to the security-key.dat file in the config directory.
- -d enables debugging.

The encryptMSAConfig Command-Line Utility

Use the encryptMSAConfig encryption command-line utility to encrypt cleartext passwords, specified by the <password> element, in XML files. Examples of XML files that can contain the <password> elements include:

- config.xml
- security-config.xml
- Component configuration files

The encryptMSAConfig utility is located in the *ORACLE_CEP_HOME*/ocep_11.1/bin directory, where *ORACLE_CEP_HOME* is the main Oracle Event Processing installation directory, such as d:\oracle_cep. The utility comes in two flavors:

- encryptMSAConfig.cmd (Windows)
- encryptMSAConfig.sh (UNIX)

encryptMSAConfig Syntax

```
encryptMSAConfig directory XML_file aesinternal.dat_file
```

where:

- *directory* refers to the directory that contains the XML file which in turn contains a cleartext <password> element.
- *XML_file* refers to the name of your XML file.
- *aesinternal.dat_file* parameter refers to the location of the .aesinternal.dat file associated with your domain; this file is located in the *ORACLE_CEP_HOME*/user_projects/domains/*DOMAIN*/*SERVER* directory, where *ORACLE_CEP_HOME* is the main Oracle Event Processing installation directory, such as d:\oracle_cep, *DOMAIN* refers to the domain directory (such as myDomain), and *SERVER* refers to the server instance (such as myServer).

For example:

```
prompt> pwd
C:\OracleCEP\user_projects\domains\ocep_domain\defaultserver
prompt> C:\OracleCEP\ocep_11.1\bin\encryptMSAConfig.cmd . config\config.xml
.aesinternal.dat
```

After you run the command, the value of the password element in *XML_file* will be encrypted.

The GrabCert Command-Line Utility

Use the GrabCert command-line utility to generate a trust keystore that includes the certificate from an existing trust keystore.

The GrabCert utility is located in the *ORACLE_CEP_HOME*/ocep_11.1/utils/security/wlevsgrabcert.jar file, where *ORACLE_CEP_HOME* refers to the Oracle Event Processing installation directory (such as d:/oracle_cep).

GrabCert Syntax

```
java GrabCert host:secureport [-alias=alias] [-noinput] [truststorepath]
```

where:

Table C–1 GrabCert Arguments

Option	Description	Default Value
host	The host name of the Oracle Event Processing server from which to copy the certificate.	
secureport	The SSL port on *host*. For more information, see Example 10–5 in Section , "How to Configure SSL Manually."	9003
alias	The alias for the certificate in the trust keystore.	*host*
-noinput	Use the -noinput option to instruct GrabCert to copy all certificates from *host*. Omit the -noinput option to instruct GrabCert to list all available certificates from *host* and prompt you to select one.	
truststorepath	The full pathname of the generated trust keystore file on *host*.	evstrust.jks

Examples of Using GrabCert

For example:

```
prompt> java GrabCert ariel:9003 -alias=ariel evstrust.jks
```

For other examples, see Section , "How to Configure SSL in a Multi-Server Domain for Oracle Event Processing Visualizer".

The passgen Command-Line Utility

Use the passgen command-line utility to hash user passwords for addition to a security database.

> **Note:** The passgen command line utility has been deprecated as of Version 10.3 of Oracle Event Processing. This is because the Configuration Wizard automatically performs the required task for you.

The passgen utility is located in the *ORACLE_CEP_HOME*/ocep_11.1/bin directory, where *ORACLE_CEP_HOME* is the main Oracle Event Processing installation directory, such as d:\oracle_cep. The utility comes in two flavors:

- passgen.cmd (Windows)
- passgen.sh (UNIX)

passgen Syntax

```
passgen [-a algorithm] [-s saltsize] [-h] [-?] [password]*
```

where:

Table C–2 passgen Arguments

Option	Description	Default Value
`-a`	*`algorithm`* specifies the hash algorithm to use: ▪ SHA-1 ▪ MD2 ▪ MD5 ▪ SSHA ▪ SHA-256 The actual list of algorithms that can be set depends on the security providers plugged into the JDK.	If not specified, the default is SHA-1.
`-s`	*`saltsize`* is the number of salt characters added to ensure a unique hash string.	If not specified, the default is 4.
`-h, -?`	Displays command line options and exits.	
`password`	If passwords are specified on the command line they shall be hashed and printed out one per line in order from left to right. If no passwords are specified on the command line, then the tool shall prompt for passwords to hash interactively.	

Note: Windows operating systems must use the `.cmd` version of this utility, Unix platforms must use the `.sh` version.

The Unix version of this utility starts with the `#!/bin/ksh` directive. On most Unix systems, this forces the Korn Shell program to be used when using the utility. If the `ksh` program is not present in the `bin` directory or if the shell language used cannot properly execute the utility, run the utility as shown below:

```
$PATH_TO_KSH_BIN/ksh -c passgen.sh
```

where `PATH_TO_KSH_BIN` is the fully qualified path to the `ksh` program.

Examples of Using passgen

The following sections provide examples that use the `passgen` utility:

- Section , "Using passgen interactively"
- Section , "Providing a Password on the Command Line"

Using passgen interactively

The following is an example of using the `passgen` utility interactively:

```
$ passgen
Password ("quit" to end): maltese
{SHA-1}LOtYvfQZj++4rV50AKpAvwMlQjqVd7ge
Password ("quit" to end): falcon
{SHA-1}u7NPQfgkHISr0tZUsmPrPmr3U1LKcAdP
Password ("quit" to end): quit
{SHA-1}2pPo4ViKsoNct3lTDoLeg9gHYZwQ47sV
```

In this mode, a password is entered and the resulting hashed version of the password is displayed. The hashed version of the password can then be entered into the password field of a security database.

Note: In example, the passwords are shown to be echoed to the screen for demonstration purposes. In most situations, the password would not be displayed unless your platform does not support invisible passwords.

Providing a Password on the Command Line

The following is an example using the `passgen` utility when providing the passwords to be hashed on the command line:

```
$ passgen maltese falcon
{SHA-1}g0PNXmJW0OBtp/GkHrhNAhpbjM+capNe
{SHA-1}2ivZnjnKD9fordC1YFkrVGf0DHL6SVP1
```

When multiple passwords are provided, they are hashed from left to right:

- `{SHA-1}g0PNXmJW0OBtp/GkHrhNAhpbjM+capNe` is hashed from `maltese`
- `{SHA-1}2ivZnjnKD9fordC1YFkrVGf0DHL6SVP1` is hashed from `falcon`.

The secgen Command-Line Utility

Use the `secgen` command-line utility to generate a security key or a security configuration file that uses encrypted passwords.

Caution: The `secgen` command line utility has been deprecated as of Version 10.3 of Oracle Event Processing. This is because the Configuration Wizard automatically performs the required task for you.

Note: This utility creates a security file that does not use a password policy; if you require a password policy, use the `cssconfig` command-line utility instead. See Section , "The cssconfig Command-Line Utility."

The `secgen` utility is located in the *`ORACLE_CEP_HOME`*`/ocep_11.1/bin` directory, where *`ORACLE_CEP_HOME`* is the main Oracle Event Processing installation directory, such as `d:\oracle_cep`. The utility comes in two flavors:

- `secgen.cmd` (Windows)
- `secgen.sh` (UNIX)

Generating a File-Based Provider Configuration File

Use the following command line options to generate a file-based security provider configuration file.

```
secgen -F [-o outputfile] [-i inputkeyfile] [-e] [-P PropertyFilePath]
```

where:

Table C–3 secgen Arguments for a File-Based Provider Configuration File

Option	Description	Comments
`-F`	Generate a file-based security provider file; mutually exclusive with the `-k` option.	If not present, `-k` is assumed.
`-o`	*`outputfile`* is the name for the generated file.	Default output file name is `security.xml`.
`-i`	*`inputkeyfile`* is the fully qualified name of the input key file.	If not present, a default input key file named `security-key.dat` is expected.
`-e`	Enables unanimous adjudication during authorization.	
`-P`	*`PropertyFilePath`* is the fully qualified path to a `secgen` property file which you can use to customize provider configurations. See Section , "Using the secgen Properties File" for details.	A `SecGenTemplate.properties` template file is located at *`ORACLE_CEP_HOME`*`/ocep_11.1/bin` where *`ORACLE_CEP_HOME`* is the main installation directory of Oracle Event Processing, such as `/oracle_cep`.

Generating a Key File

Use the following command line options to generate a security key file.

```
secgen [-k] [-o outputfile]
```

where:

Table C–4 secgen Arguments for a Key File

Option	Description	Comments
`-k`	Generate a key file; mutually exclusive with the `-F` option.	If not present, `-k` is assumed.
`-o`	*`outputfile`* is the name for the generated file.	Default output file name is `security-key.dat`.

Using the secgen Properties File

When running secgen, you can use the `-P` option to specify a property file to customize provider configurations. A `SecGenTemplate.properties` template file is located in *`ORACLE_CEP_HOME`*`/ocep_11.1/bin` where *`ORACLE_CEP_HOME`* is the main installation directory of Oracle Event Processing, such as `/oracle_cep`.

You specify cleartext passwords the property file; however, these passwords will be stored encrypted in the generated configuration file.

The following example shows a property file used for file based provider customization:

```
#File based provider related
file.atn.file.store.path=myfileatnstore.txt
file.atn.file.store.password=firewall
file.atn.user.password.style=HASHED
file.atn.file.store.encrypted=true
file.atz.file.store.path=filatz
file.atz.file.store.password=firewall
file.rm.file.store.path=filerm
file.rm.file.store.password=firewall
file.cm.file.store.path=filecm
file.cm.file.store.password=firewall
```

The legal values for `file.atn.user.password.style` are:

- `HASHED`
- `REVERSIBLEENCRYPTED`

Examples of Using secgen

The following example shows how to use the `secgen` utility to generate a key file with the name `myKeyFile.dat`:

```
prompt> secgen -k -o myKeyFile.dat
```

The following example shows how to use the `secgen` utility to generate a file-based security provider configuration file named `myConfigFile.xml` which also uses the previously generated key file, `myKeyFile.dat`, and a properties file named `mySecGen.properties`:

```
prompt> secgen -F -i myKeyFile.dat -o myConfigFile.xml -P
c:\msa\myMSAConfig\mySecGen.properties
```

Limitations of secgen

Windows operating systems must use the `.cmd` version of this utility, Unix platforms should use the `.sh` version.

The Unix version of this utility starts with the `#!/bin/ksh` directive. On most Unix systems, this forces the Korn Shell program to be used when using the utility. If the `ksh` program is not present in the `bin` directory or if the shell language used cannot properly execute the utility, run the utility as shown below:

```
prompt> $PATH_TO_KSH_BIN/ksh -c secgen.sh
```

where *`PATH_TO_KSH_BIN`* is the fully qualified path to the `ksh` program.