BEA WebLogic® Event Server

WebLogic Event Server Administration and Configuration Guide

Version 2.0
July 2007

Contents

CHAPTER 1

Introduction and Roadmap

This section describes the contents and organization of this guide—*WebLogic Event Server Administration Guide*.

- "Document Scope and Audience" on page 1-1
- "WebLogic Event Server Documentation Set" on page 1-2
- "Guide to This Document" on page 1-2
- "Samples for the WebLogic Event Server Application Developer" on page 1-3

Document Scope and Audience

This document is a resource for software developers who develop event driven real-time applications. It also contains information that is useful for business analysts and system architects who are evaluating WebLogic Event Server or considering the use of WebLogic Event Server for a particular application.

The topics in this document are relevant during the design, development, configuration, deployment, and performance tuning phases of event driven applications. The document also includes topics that are useful in solving application problems that are discovered during test and pre-production phases of a project.

It is assumed that the reader is familiar with the Java programming language and Spring.

WebLogic Event Server Documentation Set

This document is part of a larger WebLogic Event Server documentation set that covers a comprehensive list of topics. The full documentation set includes the following documents:

- *Getting Started With WebLogic Event Server*
- *Creating WebLogic Event Server Applications*
- *WebLogic Event Server Administration and Configuration Guide*
- *EPL Reference Guide*
- *WebLogic Event Server Reference Guide*
- *WebLogic Event Server Release Notes*

See the main WebLogic Event Server documentation page for further details.

Guide to This Document

This document is organized as follows:

- This chapter, Chapter 1, "Introduction and Roadmap," introduces the organization of this guide and the WebLogic Event Server documentation set and samples.
- Chapter 2, "Creating a WebLogic Event Server Domain," describes how to create a new WebLogic Event Server domain.
- Chapter 3, "Configuring WebLogic Event Server," describes how to start and stop a WebLogic Event Server server, as well as how to configure it.
- Chapter 5, "wlevs.Admin Command-Line Reference," provides reference information about using the `wlevs.Admin` utility to configure WebLogic Event Server and the EPL rules attached to a particular application.
- Chapter 6, "Configuring Security for WebLogic Event Server," provides information about configuring various types of security for WebLogic Event Server.
- Chapter 7, "Configuring Jetty for WebLogic Event Server," describes how to configure some features of the microServices Architecture, in particular the Jetty Web Service, work managers, and net IO.
- Chapter 8, "Configuring JMX for WebLogic Event Server," describes how to configure some features of the microServices Architecture, in particular RMI, JNDI, and JMX.

- Chapter 9, "Configuring Access to a Relational Database," describes how to configure access to a relational database.
- Chapter 10, "Configuring Logging and Debugging," describes how to configure the logging and debugging features of the microServices Architecture.

Samples for the WebLogic Event Server Application Developer

In addition to this document, BEA Systems provides a variety of code samples for WebLogic Event Server application developers. The examples illustrate WebLogic Event Server in action, and provide practical instructions on how to perform key development tasks.

BEA recommends that you run some or all of the examples before programming and configuring your own event driven application.

The examples are distributed in two ways:

- Pre-packaged and compiled in their own domain so you can immediately run them after you install the product.
- Separately in a Java source directory so you can see a typical development environment setup.

The following two examples are provided in both their own domain and as Java source in this release of WebLogic Event Server:

- HelloWorld—Example that shows the basic elements of a WebLogic Event Server application. See Hello World Example for additional information.

  The HelloWorld domain is located in `WLEVS_HOME\samples\domains\helloworld_domain`, where `WLEVS_HOME` refers to the top-level WebLogic Event Server directory, such as `c:\beahome\wlevs20`.

  The HelloWorld Java source code is located in `WLEVS_HOME\samples\source\applications\helloworld`.

- ForeignExchange (FX)—Example that includes multiple adapters, streams, and complex event processor with a variety of EPL rules, all packaged in the same WebLogic Event Server application. See Foreign Exchange (FX) Example for additional information.

  The ForeignExchange domain is located in `WLEVS_HOME\samples\domains\fx_domain`, where `WLEVS_HOME` refers to the top-level WebLogic Event Server directory, such as `c:\beahome\wlevs20`.

The ForeignExchange Java source code is located in `WLEVS_HOME\samples\source\applications\fx`.

WebLogic Event Server also includes an algorithmic trading application, pre-assembled and deployed in its own sample domain; the source code for the example, however, is not provided. The algorithmic trading domain is located in `WLEVS_HOME\samples\domains\algotrading_domain`.

CHAPTER 2

Creating a WebLogic Event Server Domain

This section contains information on the following subjects:

Overview of WebLogic Event Server Domains

A *domain* is the basic administration unit for WebLogic Event Server. It consists of one WebLogic Event Server instance, zero or more deployed applications, and logically related resources and services that are managed, collectively, as one unit.

After you install WebLogic Event Server, you can use the empty domain called `*BEA_HOME*/user_projects/domains/wlevs20_domain` to deploy your applications, or you can use it as a template to help you create your own domain, where `*BEA_HOME*` refers to the parent directory of the main WebLogic Event Server installation directory such as `d:/beahome`.

The following list describes the important files and directories in a domain, relative to the main domain directory:

- `deployments.xml`—XML file that contains the list of applications, packaged as OSGi bundles, that are currently deployed to the WebLogic Event Server instance of this domain.

- `startwlevs.cmd`—Command file used to start an instance of WebLogic Event Server. The UNIX equivalent is called `startwlevs.sh`.
- `stopwlevs.cmd`—Command file used to stop an instance of WebLogic Event Server. The UNIX equivalent is called `stopwlevs.sh`.
- `config/config.xml`—XML file that describes the services that have been configured for the WebLogic Event Server instance. Services include logging, debugging, Jetty Web Service, and JDBC data sources.
- `applications`—Default directory that contains the OSGi bundles and XML configuration files for the deployed applications, or those that are listed in the `deployments.xml` file. Each application lives in its own subdirectory of the applications directory.
- `config/security*`—Files that configure security for the domain.

Creating a Domain

Creating a WebLogic Event Server domain entails manually copying the sample empty domain to a new directory. In particular, copy the directory that contains the sample domain, called `wlevs20_domain`, to a new directory under the *`BEA_HOME`*`\user_projects\domains` directory. Name the new directory the same as the name of your new domain.

For example, if you installed WebLogic Event Server in the `d:\beahome2\wlevs20` directory and want to create a new domain called `myDomain`, the copy command would be:

```
prompt> cd d:\beahome2\user_projects\domains

prompt> cp -r wlevs20_domain myDomain
```

Stopping and Starting the Server

Each WebLogic Event Server domain contains a command script that starts a server instance; by default, the script is called `startwlevs.cmd` (Windows) or `startwlevs.sh` (UNIX). The script to stop the server is called `stopwlevs.cmd` (Windows) or `stopwlevs.sh` (UNIX).

Starting the Server

To start an instance of WebLogic Event Server:

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

The server start script is located in the main domain directory. For example, the HelloWorld domain is located in `WLEVS_HOME/samples/domains/helloworld_domain`, where `WLEVS_HOME` refers to the main WebLogic Event Server installation directory, such as `/beahome_wlevs/wlevs20`.

If using the JRockit JDK installed with WebLogic Real Time 2.0, the `JAVA_HOME` variable should be set as follows:

```
JAVA_HOME=BEA_HOME_WLRT/jrockit-realtime20_150_11 (UNIX)
set JAVA_HOME=BEA_HOME_WLRT\jrockit-realtime20_150_11 (Windows)
```

where `BEA_HOME_WLRT` refers to the installation directory of WebLogic Real Time 2.0, such as `/beahome_wlrt` (UNIX) or `c:\beahome_wlrt` (Windows).

If using the JRockit JDK installed with WebLogic Event Server 2.0, the `JAVA_HOME` variable should be set as follows:

```
JAVA_HOME=BEA_HOME_WLEVS/jrockit-R27.3.0-106-1.5.0_11 (UNIX)
set JAVA_HOME=BEA_HOME_WLEVS\jrockit-R27.3.0-106-1.5.0_11 (Windows)
```

where `BEA_HOME_WLEVS` refers to the installation directory of WebLogic Event Server 2.0, such as `/beahome_wlevs` (UNIX) or `c:\beahome_wlevs` (Windows).

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

   ```
   prompt> cd C:\bea_wlevs\wlevs20\samples\domains\helloworld_domain
   ```

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

   ```
   prompt> startwlevs.cmd
   ```

   If you are using the JRockit JDK included in WebLogic Real Time 2.0, enable the deterministic garbage collector by passing the `-dgc` parameter to the command:

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

Stopping the Server Using the stopwlevs Script

To stop a running WebLogic Event Server:

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

   ```
   prompt> cd C:\bea_wlevs\wlevs20\samples\domains\helloworld_domain
   ```

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:rmi:///jndi/rmi://`*`host:jmxport/`*`jmxrmi`, where *`host`* refers to the computer hosting the server and *`jmxport`* refers to the server's JMX port, configured in `config.xml` file. For example:

```
prompt> stopwlevs.sh -url
service:jmx:rmi:///jndi/rmi://ariel:1099/jmxrmi
```

In the example, the host is `ariel` and the JMX port is `1099`.

See Table 5-1, "Connection Arguments," on page 5-6 for additional details about the `-url` argument.

Next Steps

After creating your own WebLogic Event Server domain:

- Optionally configure the server and domain. See "Overview of Configuring WebLogic Event Server" on page 3-1.
- Create a WebLogic Event Server application. See Creating WebLogic Event Server Applications for a description of the programming model, details about the various components that make up an application, and how they all fit together.
- Deploy your new, or existing, WebLogic Event Server application to the domain. See Deploying WebLogic Event Server Applications.

CHAPTER 3

Configuring WebLogic Event Server

This section contains information on the following subjects:

- “Overview of Configuring WebLogic Event Server” on page 3-1
- “Configuring the Server by Manually Editing the config.xml File” on page 3-2

Overview of Configuring WebLogic Event Server

After you have created a WebLogic Event Server domain, you start a server instance so you can then deploy applications and begin running them. See “Stopping and Starting the Server” on page 2-2 for details.

There are a variety of ways to configure a particular server instance, as follows:

- Update the server configuration file, config.xml, manually.

  See “Configuring the Server by Manually Editing the config.xml File” on page 3-2.

- Use the `wlevs.Admin` utility to administer WebLogic Event Server and to dynamically configure the EPL rules for the processors of a deployed application.

  See “wlevs.Admin Command-Line Reference” on page 5-1.

- Use standards-based interfaces that are fully compliant with the Java Management Extensions (JMX) specification to change the configuration of the domain and deployed applications.

  See “Configuring Applications and Servers Dynamically” on page 4-1 and the Javadoc for details about the WebLogic Event Server MBeans.

Configuring the Server by Manually Editing the config.xml File

The WebLogic Event Server configuration file, `config.xml`, is located in the *`DOMAIN_DIR`*`/config` directory, where *`DOMAIN_DIR`* refers to the main domain directory. To change the configuration of WebLogic Event Server, you can update this file manually and add or remove server configuration elements.

In this release of WebLogic Event Server, the `config.xml` file configures:

- Logging and debugging properties of the server. By default, the log security level is set to `NOTICE`.

  See "Configuring Logging and Debugging" on page 10-1.

- Jetty, an open-source, standards-based, full-featured Java Web Server.

  See "Configuring Jetty for WebLogic Event Server" on page 7-1.

- JDBC data source, used to connect to a relational database.

  See "Configuring Access to a Relational Database" on page 9-1.

- JMX, required to use the `wlevs.Admin` utility.

  See "Configuring JMX for WebLogic Event Server" on page 8-1.

The following sample `config.xml`, from the *`BEA_HOME`*`/user_projects/domains/wlevs20_domain` template domain, shows how to configure some of these services:

```
<n1:config
xsi:schemaLocation="http://www.bea.com/wlevs/xml/ns/config/server
wlevs_server_config.xsd"
xmlns:n1="http://www.bea.com/wlevs/xml/ns/config/server"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">

  <netio>
    <name>JettyNetIO</name>
    <port>9002</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>JettyNetIO</network-io-name>
    <work-manager-name>JettyWorkManager</work-manager-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>
    <jndi-service-name>JNDI</jndi-service-name>
    <rmi-service-name>RMI</rmi-service-name>
    <rmi-registry-port>1099</rmi-registry-port>
    <rmi-jrmp-port>9999</rmi-jrmp-port>
  </jmx>

</n1:config>
```

WARNING: If you update the `config.xml` file manually to change the configuration of WebLogic Event Server, you must restart the server for the change to take effect.

CHAPTER 4

Configuring Applications and Servers Dynamically

This section contains information on the following subjects:

Overview of Dynamic Configuration

WebLogic Event Server applications define an event processing network (EPN) that is made up of components such as processors, streams, and adapters. You deploy these applications to a WebLogic Event Server 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 WebLogic Event Server documentation, this section uses the term *components*.

You can dynamically configure each componnent in the EPN using managed beans, or *MBeans*. You manipulate the MBeans either by using the standard Java Management Extenstion (JMX) APIs or using `wlevs.Admin`, the WebLogic Event Server administration command-line utility. It is assumed in this section you are going to use JMX; see "wlevs.Admin Command-Line Reference" on page 5-1 for details about using `wlevs.Admin`.

You can also perform some configuration and application lifecycle 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, stream, 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 stream or the EPL rules associated with a processor. You can also dynamically enable or disable monitoring for a component.

Overview of WebLogic Event Server MBeans

WebLogic Event Server exposes the following types of MBeans:

- **Standard Configuration MBeans**—Contain information about the configuration of components in an EPN, a deployed WebLogic Event Server 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 `EPLProcessorMBean` and `StreamMBean`.

  See "Configuration MBeans" on page 4-3 for additional information.

- **Custom Configuration Extension MBeans**—Similar to standard configuration MBeans, but correspond to adapters whose configuration has been extended by a programmer. For example, the `helloworldAdapter` adapter in the HelloWorld example has been extended, and its corresponding MBean would be `HelloWorldAdapterConfigMBean`.

  See "Configuration MBeans" on page 4-3 for additional information.

For full reference information about WebLogic Event Server MBeans, see the following Javadocs:

- `com.bea.wlevs.management.configuration`
- `com.bea.wlevs.management.boot`
- `com.bea.wlevs.management`
- `com.bea.wlevs.server.management.mbean`

Configuration MBeans

When you deploy a WebLogic Event Server 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 whose `manageable` attribute is set to `true`. If you have extended the configuration of an adapter, then the server deploys a 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 stream component. A common configuration method for all components is to enable or disable monitoring; for example, to disable monitoring for an adapter, use the method `AdapterMBean.disableMonitoring()`.

MBean Hierarchy and Naming

All MBeans must be registered in an MBean server under an object name of type `javax.management.ObjectName`. WebLogic Event Server follows a convention in which object names for child MBeans contain part of its parent MBean object name.

Configuration MBean Naming

WebLogic Event Server 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` is as follows:

```
com.bea.wlevs:Name=myprocessor,Type=EPLProcessor,Application=myapplication
```

The following table describes the key properties that WebLogic Server encodes in its MBean object names.

Table 4-1 WebLogic Event Server 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"`
	`manageable="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 `EPLProcessorMBean`, use `EPLProcessor`. In this case, the key property would be `Type=EPLProcessor`.

`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 WebLogic Event Server 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`.

The following table 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`

Table 4-2 Component Declaration Example With Corresponding MBean Object Names

Sample Component Declaration in EPN Assembly File	Corresponding Configuration MBean Object Name
`<wlevs:adapter id="helloworldAdapter" provider="hellomsgs" manageable="true">` `<wlevs:instance-property name="message" value="HelloWorld - the current time is:"/>` `</wlevs:adapter>`	`com.bea.wlevs:Name=helloworldAdapter,Type=HelloworldAdapterConfig,Application=helloworld` `HelloworldAdapterConfig` is an example of a custom configuration extension MBean. Its manageable property is `message`.
`<wlevs:processor id="helloworldProcessor" manageable="true" />`	`com.bea.wlevs:Name=helloworldProcessor,Type=EPLProcessor,Application=helloworld` `EPLProcessor` is the standard configuration MBean for processor components. The manageable property is `rules`.
`<wlevs:stream id="helloworldInstream" manageable="true">` `<wlevs:listener ref="helloworldProcessor"/>` `<wlevs:source ref="helloworldAdapter"/>` `</wlevs:stream>`	`com.bea.wlevs:Name=helloworldInstream,Type=Stream,Application=helloworld` `Stream` is the standard configuration MBean for a stream component. The manageable properties are `MaxSize` and `MaxThreads`.

Dynamically Configuring a Component Using JMX: Typical Steps

It is assumed in this section that you are going to use the Java Management Extensions (JMX) APIs to manipulate the configuration MBeans; if you want to use `wlevs.Admin`, see Chapter 5, “wlevs.Admin Command-Line Reference.” Be sure you have read the following sections that describe WebLogic Event Server configuration and runtime MBeans:

- “Overview of Dynamic Configuration” on page 4-1
- “Overview of WebLogic Event Server MBeans” on page 4-2
- “MBean Hierarchy and Naming” on page 4-3

To dynamically configure a component of an EPN, follow these steps:

1. Be sure that the JMX service is configured for your domain. For details see Chapter 8, “Configuring JMX for WebLogic Event Server.”
2. Be sure that the **`manageable`** attribute of the component in the EPN assembly file has been set to `true`. By default the attribute is `false`. For example:

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

   You cannot dynamically change the value of this attribute. In practice this means that, for example, if you have set `manageable` to `true` and deployed the application, but then you later want to disable manageability, you must undeploy the application, manually update the EPN assembly file and set `manageable` to `false`, then redeploy the application.
3. Write the JMX Java code to configure the component using the appropriate MBean.

   For example, assume you want to configure a processor called `helloworldProcessor` that is part of the WebLogic Event Server application called `myapplication`. The following Java code shows how to instantiate an instance of the appropriate MBean (`EPLProcessorMBean`) and then get a list of all the EPL rules associated with the processor:

   ```
   ObjectName eplName =

   ObjectName.getInstance("com.bea.wlevs:Name=helloworldProcessor,Type=EPL
   Processor,Application=myapplication");

   EPLProcessorMBean eplMBean = (EPLProcessorMBean)
              MBeanServerInvocationHandler.newProxyInstance(
                                   mbsc,
                                   ObjectName.getInstance(eplName),
                                   EPLProcessorMBean.class,
                                   true);

   Map rules =  eplMBean.getAllRules();
   ```

CHAPTER 5

wlevs.Admin Command-Line Reference

The following sections describe the `wlevs.Admin` utility:

Overview of the wlevs.Admin Utility

The `wlevs.Admin` utility is a command-line interface to administer WebLogic Event Server and, in particular, dynamically configure the EPL rules for application processors. The utility internally uses JMX to query the configuration and runtime MBeans of both the server and deployed applications.

The WebLogic Event Server configuration framework allows concurrent changes to both the application and 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 command of the `wlevs.Admin` utility 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.

Required Environment for the wlevs.Admin Utility

To set up your environment for the `wlevs.Admin` utility:

1. Install and configure the WebLogic Event Server software, as described in the WebLogic Event Server *Installation Guide*.
2. Configure JMX connectivity for the domain you want to administer. See "Configuring JMX for WebLogic Event Server" on page 8-1.
3. Open a command window and set your environment as described in Setting Up Your Development Environment.
4. 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 `*BEA_HOME*/wlevs20/bin/com.bea.wlevs.management.commandline_2.0.0.0.jar` file, where `*BEA_HOME*` refers to the main BEA Home directory into which you installed WebLogic Event Server. 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:/beahome/wlevs20/bin/com.bea.wlevs.management.commandline_2.0.0.0.
     jar
     -url service:jmx:rmi:///jndi/rmi://localhost:1099/jmxrmi
     UPLOAD -application helloworld -processor helloworldProcessor
     -sourceURL file:///d:/test/newrules2.xml
     ```
   - Explicitly update your CLASSPATH by adding the following files to the CLASSPATH environment variable:

- *BEA_HOME*/wlevs20/bin/com.bea.wlevs.management.commandline_2.0.0.0.jar
- *BEA_HOME*/wlevs20/bin/wlevs_2.0.jar
- *BEA_HOME*/wlevs20/modules/com.bea.wlevs.deployment.server_2.0.0.0.jar
- *BEA_HOME*/wlevs20/modules/com.bea.wlevs.ede_2.0.0.0.jar
- *BEA_HOME*/wlevs20/modules/com.bea.wlevs.management_2.0.0.0.jar
- *BEA_HOME*/modules/com.bea.core.jmx_3.0.0.0.jar
- *BEA_HOME*/modules/com.bea.core.jndi.context_3.0.0.0.jar
- *BEA_HOME*/modules/com.bea.core.rmi_3.0.0.0.jar
- *BEA_HOME*/modules/com.bea.core.i18n_1.1.0.0.jar
- *BEA_HOME*/modules/modules/com.bea.core.i18n.generator_1.1.0.0.jar
- *BEA_HOME*/modules/com.bea.core.diagnostics.core_1.1.0.0.jar
- *BEA_HOME*/modules/javax.xml.stream_1.1.0.0.jar

where *BEA_HOME* refers to the main BEA Home directory into which you installed WebLogic Event Server.

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 WebLogic Event Server is installed and running. To run the utility remotely, follow these steps:

1. Copy the following JAR files from the computer on which WebLogic Event Server 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:
   - *BEA_HOME*/wlevs20/bin/com.bea.wlevs.management.commandline_2.0.0.0.jar
   - *BEA_HOME*/wlevs20/modules/com.bea.wlevs.deployment.server_2.0.0.0.jar
   - *BEA_HOME*/wlevs20/modules/com.bea.wlevs.ede_2.0.0.0.jar
   - *BEA_HOME*/wlevs20/modules/com.bea.wlevs.management_2.0.0.0.jar
   - *BEA_HOME*/modules/com.bea.core.jmx_3.0.0.0.jar
   - *BEA_HOME*/modules/com.bea.core.jndi.context_3.0.0.0.jar

  – *BEA_HOME*`/modules/com.bea.core.rmi_3.0.0.0.jar`
  – *BEA_HOME*`/modules/com.bea.core.i18n_1.1.0.0.jar`
  – *BEA_HOME*`/modules/modules/com.bea.core.i18n.generator_1.1.0.0.jar`
  – *BEA_HOME*`/modules/com.bea.core.diagnostics.core_1.1.0.0.jar`
  – *BEA_HOME*`/modules/javax.xml.stream_1.1.0.0.jar`

  where *BEA_HOME* refers to the main BEA Home directory into which you installed WebLogic Event Server.

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/*`com.bea.wlevs.management.commandline_2.0.0.0.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.

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:

- "Connection Arguments" on page 5-5
- "User Credentials Arguments" on page 5-6
- "Common Arguments" on page 5-7

The following sections provide detailed syntax information about the supported commands of the `wlevs.Admin` utility:

- "Commands for Managing the Server Life Cycle" on page 5-9
- "Commands for Managing the EPL Rules of an Application" on page 5-10
- "Commands for Managing WebLogic Event Server MBeans" on page 5-19

Example Environment

In many of the examples throughout the sections that follow, it is assumed that a certain environment has been set up:

- The WebLogic Event Server instance listens to JMX requests on port `1099`.
- The WebLogic Event Server instance uses the name of its host machine, `ariel`, as its listen address.
- The `wleves` username has system-administrator privileges and uses `wlevs` for a password.

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} ]
     [ User Credentials Arguments ]
     [ Common Arguments ]
     COMMAND-NAME command-arguments
```

When you invoke most `wlevs.Admin` commands, you specify the arguments in Table 5-1 to connect to a WebLogic Event Server instance.

Table 5-1 Connection Arguments

Argument	Definition

`-url service:jmx:rmi:///jndi/rmi://`*`host`*`:`*`jmxport`*`/jmxrmi`	Specifies the URL that establishes a JMX connection to the WebLogic Event Server instance you want to administer, where:
	• *`host`* refers to the name of the computer on which the WebLogic Event Server instance is running.
	• *`jmxport`* refers to the port configured for WebLogic Event Server that listens to JMX connections.
	This port is configured in the `config.xml` file of the WebLogic Event Server domain you are administering. In particular, you specify the port using the `<rmi-registry-port>` child element of the `<jmx>` element, as shown:
	`<jmx>`
	`    <jndi-service-name>JNDI</jndi-service-name>`
	`    <rmi-service-name>RMI</rmi-service-name>`
	`    <rmi-registry-port>1099</rmi-registry-port>`
	`    <rmi-jrmp-port>9999</rmi-jrmp-port>`
	`</jmx>`
	In the example, the JMX port is `1099`.
	Other than *`host`* and *`jmxport`*, you specify the remainder of the URL as written.
	For example, if WebLogic Event Server is running on a computer with hostname ariel, and the JMX listening port is 1099, then the URL would be:
	`-url service:jmx:rmi:///jndi/rmi://ariel:1099/jmxrmi`
	See "Configuring JMX for WebLogic Event Server" on page 8-1 for details about configuring JMX, JNDI, and RMI for WebLogic Event Server.

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 5-2 to provide the user credentials of a WebLogic Event Server user who has permission to invoke the command.

If security has not been enabled for your WebLogic Event Server domain, then you do not have to provide user credentials.

Table 5-2 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 WebLogic Event Server 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 5-3 to get verbose output.

Table 5-3 Common Arguments

Argument	Definition
`-verbose`	Specifies that `wlevs.Admin` should output additional verbose information.

Command for Getting Usage Help

HELP

Provides syntax and usage information for all WebLogic Event Server 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 WebLogic Event Server 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 ULOAD
```

The command returns the following:

```
Description:
Uploads rules to be configured in the EPL Processor.

Usage:
java wlevs.Admin
   [-url | -listenAddress <host-name> -listenPort <port>]
  -username <username> -password <password>
 UPLOAD -application <application name> -processor <eplprocessor name>
-sourceURL "source url"

Where:
-application = Name of the application.
-processor = Name of the EPL Processor.
-sourceURL = source URL containing the rules in an XML format.

java wlevs.Admin -url service:jmx:rmi:///jndi/rmi://localhost:1099/jmxrmi
-username wlevs -password wlevs UPLOAD -application myapplication -processor
eplprocessor -sourceURL file:///d:/test/rules.xml
```

Commands for Managing the Server Life Cycle

Table 5-4 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 5-4 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 WebLogic Event Server instance.

A graceful shutdown gives WebLogic Event Server time to complete certain application processing currently in progress.

The `-url` connection argument specifies the particular WebLogic Event Server instance that you want to shut down, based on the `host` and `jmxport` values. See "Connection Arguments" on page 5-5 for details.

Syntax

```
java wlevs.Admin
     [ Connection Arguments ]
     [ User Credentials Arguments ]
     [ Common Arguments ]
     SHUTDOWN [-scheduleAt seconds]
```

Table 5-5 SHUTDOWN Arguments

Argument	Definition
`-scheduleAt` *`seconds`*	Specifies the number of seconds after which the WebLogic Event Server instance shuts down. If you do not specify this parameter, the server instance shuts down immediately.

Example

The following example instructs the specified WebLogic Event Server instance to shut down in ten minutes:

```
prompt> java wlevs.Admin
       -url service:jmx:rmi:///jndi/rmi://ariel:1099/jmxrmi
       -username wlevs -password wlevs
        SHUTDOWN -scheduleAt 600
```

Note: For clarity, the preceding example is shown on multiple lines; however, when you run the command, enter all arguments and commands on a single line.

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 EPL Rules of an Application

Table 5-4 is an overview of commands that manage the EPL rules for a particular processor of a WebLogic Event Server application. Subsequent sections describe command syntax and arguments, and provide an example for each command.

Table 5-6 Overview of Commands for Managing Application EPL Rules

Command	Description
ADDRULE	Adds a new EPL rule to the processor of a WebLogic Event Server application.
DELETERULE	Deletes an existing EPL rule from the processor of a WebLogic Event Server application.
GETRULE	Returns the text of an existing EPL rule of the processor of a WebLogic Event Server application.
UPLOAD	Configures a set of EPL rules for a processor of a WebLogic Event Server application by uploading the rules from an XML file.
DOWNLOAD	Downloads the set of EPL rules associated with a processor of a WebLogic Event Server application to a file.

ADDRULE

Adds a new EPL rule to the specified processor of a WebLogic Event Server application.

Syntax

```
java wlevs.Admin
      [ Connection Arguments ]
      [ User Credentials Arguments ]
      [ Common Arguments ]
     ADDRULE -application application -processor processor -rule [rulename]
rulestring
```

Table 5-7 ADDRULE Arguments

Argument	Definition
`-application` *`application`*	Specifies the name of the WebLogic Event Server application whose EPL rules you want to manage. See "Querying for Application and Processor Names" on page 5-26 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 WebLogic Event Server application specified with the `-application` argument, whose EPL rules you want to manage. See "Querying for Application and Processor Names" on page 5-26 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, WebLogic Event Server generates a name for you. Enter the EPL rules 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 deployed to the specified WebLogic Event Server application:

```
prompt> java wlevs.Admin
        -url service:jmx:rmi:///jndi/rmi://ariel:1099/jmxrmi
        -username wlevs -password wlevs
        ADDRULE –application helloworld –processor helloworldProcessor
        –rule myrule "SELECT * FROM Withdrawal RETAIN 5 EVENTS"
```

Note: For clarity, the preceding example is shown on multiple lines; however, when you run the command, enter all arguments and commands on a single line.

DELETERULE

Deletes an existing EPL rule from the specified processor of a WebLogic Event Server application.

Syntax

```
java wlevs.Admin
     [ Connection Arguments ]
     [ User Credentials Arguments ]
     [ Common Arguments ]
    DELETERULE -application application -processor processor -rule rulename
```

Table 5-8 DELETERULE Arguments

Argument	Definition
`-application` *`application`*	Specifies the name of the WebLogic Event Server application whose EPL rules you want to manage. See "Querying for Application and Processor Names" on page 5-26 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 WebLogic Event Server application specified with the `-application` argument, whose EPL rules you want to manage. See "Querying for Application and Processor Names" on page 5-26 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 "Querying for Application and Processor Names" on page 5-26 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 deployed to the specified WebLogic Event Server application:

```
prompt> java wlevs.Admin
        -url service:jmx:rmi:///jndi/rmi://ariel:1099/jmxrmi
        -username wlevs -password wlevs
        DELETERULE –application helloworld –processor helloworldProcessor
        –rule myrule
```

Note: For clarity, the preceding example is shown on multiple lines; however, when you run the command, enter all arguments and commands on a single line.

GETRULE

Returns the full text of an EPL rule from the specified processor of a WebLogic Event Server application.

Syntax

```
java wlevs.Admin
     [ Connection Arguments ]
     [ User Credentials Arguments ]
     [ Common Arguments ]
     GETRULE -application application -processor processor -rule rulename
```

Table 5-9 GETRULE Arguments

Argument	Definition
`-application` *`application`*	Specifies the name of the WebLogic Event Server application whose EPL rules you want to manage. See "Querying for Application and Processor Names" on page 5-26 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 WebLogic Event Server application specified with the `-application` argument, whose EPL rules you want to manage. See "Querying for Application and Processor Names" on page 5-26 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 "Querying for Application and Processor Names" on page 5-26 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 deployed to the specified WebLogic Event Server application:

```
prompt> java wlevs.Admin
        -url service:jmx:rmi:///jndi/rmi://ariel:1099/jmxrmi
        -username wlevs -password wlevs
        GETRULE –application helloworld –processor helloworldProcessor
        –rule myrule
```

Note: For clarity, the preceding example is shown on multiple lines; however, when you run the command, enter all arguments and commands on a single line.

UPLOAD

Replaces the configured EPL rules for a specified processer with the EPL rules from an uploaded XML file.

The XML file that contains the list of EPL rules conforms to the processor configuration XSD Schema. 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`.

WARNING: 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 the Complex Event Processor Rules 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 5-10 UPLOAD Arguments

Argument	Definition
`-application` *`application`*	Specifies the name of the WebLogic Event Server application whose EPL rules you want to manage. See "Querying for Application and Processor Names" on page 5-26 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 WebLogic Event Server application specified with the `-application` argument, whose EPL rules you want to manage. See "Querying for Application and Processor Names" on page 5-26 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 deployed to the specified WebLogic Event Server application:

```
prompt> java wlevs.Admin
        -url service:jmx:rmi:///jndi/rmi://ariel:1099/jmxrmi
        -username wlevs -password wlevs
        UPLOAD –application helloworld –processor helloworldProcessor
        –sourceURL file:///c:/processor/config/myrules.xml
```

Note: For clarity, the preceding example is shown on multiple lines; however, when you run the command, enter all arguments and commands on a single line.

DOWNLOAD

Downloads the set of EPL rules associated with the specified processor of a WebLogic Event Server application to an XML file.

The XML file is of the same format as described in "UPLOAD" on page 5-15.

Syntax

```
java wlevs.Admin
     [ Connection Arguments ]
     [ User Credentials Arguments ]
     [ Common Arguments ]
     DOWNLOAD -application application -processor processor
     -file destinationfile [-overwrite overwrite]
```

Table 5-11 DOWNLOAD Arguments

Argument	Definition
`-application` *`application`*	Specifies the name of the WebLogic Event Server application whose EPL rules you want to manage. See "Querying for Application and Processor Names" on page 5-26 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 WebLogic Event Server application specified with the `-application` argument, whose EPL rules you want to manage. See "Querying for Application and Processor Names" on page 5-26 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:rmi:///jndi/rmi://ariel:1099/jmxrmi
        -username wlevs -password wlevs
        DOWNLOAD –application helloworld –processor helloworldProcessor
        –file c:\processor\config\myrules.xml
```

Note: For clarity, the preceding example is shown on multiple lines; however, when you run the command, enter all arguments and commands on a single line.

Commands for Managing WebLogic Event Server MBeans

The following sections describe `wlevs.Admin` commands for managing WebLogic Event Server MBeans.

- "Specifying MBean Types" on page 5-19
- "MBean Management Commands" on page 5-19

See the Javadoc for the full description of the WebLogic Event Server 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 WebLogic Event Server 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 WebLogic Event Server MBean interface classes, see the Javadocs 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.EPLProcessorMBean`, use `EPLProcessor`.

MBean Management Commands

Table 5-12 is an overview of the MBean management commands.

Table 5-12 MBean Management Command Overview

Command	Description
GET	Displays properties of MBeans.
INVOKE	Invokes management operations that an MBean exposes for its underlying resource.

Table 5-12 MBean Management Command Overview (Continued)

Command	Description
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 `javax.management.ObjectName` 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 5-13 GET Arguments

Argument	Definition
`-type` *`mbeanType`*	Returns information for all MBeans of the specified type. For more information, see "Specifying MBean Types" on page 5-19.
`-mbean` *`objectName`*	Fully qualified object name of an MBean in the `javax.management.ObjectName` format. For example, if you want to look up 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>"`
`-pretty`	Places property-value pairs on separate lines.
`-property` *`property`*	The name of the MBean property (attribute) or properties to be listed. **Note:** If property is not specified using this argument, all properties are displayed.

Example

The following example displays all properties of the `EPLProcessorMBean` that was registered for the Processor Stage when the application called `helloworld` was deployed in Weblogic Event Server.

```
prompt> java wlevs.Admin
         -url service:jmx:rmi:///jndi/rmi://ariel:1099/jmxrmi
         -username wlevs -password wlevs
         GET -pretty
         -mbean
com.bea.wlevs:Name=eplprocessor,Type=EPLProcessor,Application=helloworld
```

Note: For clarity, the preceding example is shown on multiple lines; however, when you run the command, enter all arguments and commands on a single line.

For more information about the environment in which this example runs, see "Example Environment" on page 5-5.

The following example displays all instances of all `EPLProcessorMBean` MBeans.

```
prompt> java wlevs.Admin
        -url service:jmx:rmi:///jndi/rmi://ariel:1099/jmxrmi
        -username wlevs -password wlevs
        GET -pretty -type EPLProcessor
```

INVOKE

Invokes a management operation for one or more MBeans. For WebLogic Event Server MBeans, you usually use this command to invoke operations other than the `get`*`Attribute`* and `set`*`Attribute`* that most WebLogic Event Server MBeans provide.

Syntax

```
java wlevs.Admin
     [ Connection Arguments ]
     [ User Credentials Arguments ]
     [ Common Arguments ]
    INVOKE {–type mbeanType|–mbean objectName} –method methodname [argument
. . .]
```

Table 5-14 INVOKE Arguments

Arguments	Definition
`-type` *`mbeanType`*	Invokes the operation on all MBeans of a specific type. For more information, see “Specifying MBean Types” on page 5-19.
`-mbean` *`objectName`*	Fully qualified object name of an MBean in the `javax.management.ObjectName` 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>"`

Table 5-14 INVOKE Arguments

Arguments	Definition
`-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 the `com.bea.wlevs.configuration.application.DefaultProcessorConfig` MBean:

```
prompt> java wlevs.Admin
        -url service:jmx:rmi:///jndi/rmi://ariel:1099/jmxrmi
        -username wlevs -password wlevs
        INVOKE -mbean
com.bea.wlevs:Name=eplprocessor,Type=EPLProcessor,Application=helloworld
        -method addRule "SELECT * FROM Withdrawal RETAIN ALL"
```

Note: For clarity, the preceding example is shown on multiple lines; however, when you run the command, enter all arguments and commands on a single line.

For more information about the environment in which this example runs, see "Example Environment" on page 5-5.

QUERY

Searches for WebLogic Event Server MBeans whose `javax.management.ObjectName` matches a pattern that you specify.

All MBeans that are created from a WebLogic Event Server MBean type are registered in the MBean Server under a name that conforms to the `javax.management.ObjectName` 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 5-15 QUERY Arguments

Argument	Definition
`-pretty`	Places property-value pairs on separate lines.
`-pattern` *`object-name-pattern`*	A partial `javax.management.ObjectName` 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 `javax.management.ObjectName`. 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.configuration.application.DefaultProcessorConfig` MBeans:

```
prompt> java wlevs.Admin
        -url service:jmx:rmi:///jndi/rmi://ariel:1099/jmxrmi
        -username wlevs -password wlevs
         QUERY -pattern *:Type=EPLProcessor,*
```

Note: For clarity, the preceding example is shown on multiple lines; however, when you run the command, enter all arguments and commands on a single line.

If the command succeeds, it returns the following:

```
Ok
```

For more information about the environment in which this example runs, see "Example Environment" on page 5-5.

Querying for Application and Processor Names

All the commands for managing the EPL rules of a WebLogic Event Server 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 WebLogic Event Server instance:

```
  -pattern  com.bea.wlevs:*,Type=EPLProcessor
```

For example:

```
prompt> java wlevs.Admin -url
        service:jmx:rmi:///jndi/rmi://ariel:1099/jmxrmi
        -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=hellowor
ld,"
       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 5-16 SET Arguments

Argument	Definition
`-type` *`mbeanType`*	Sets the properties for all MBeans of a specific type. For more information, see "Specifying MBean Types" on page 5-19.
`-mbean` *`objectName`*	Fully qualified object name of an MBean in the `javax.management.ObjectName` format: `"com.bea.wlevs:Name=<name of the stage>,Type=<MBean type>,Application=<name of the deployed application>"`

Table 5-16 SET Arguments

Argument	Definition
`-property` *`property`*	The name of the property to be set.

`property _value`	The value to be set.
	• Some properties require you to specify the name of a WebLogic Event Server MBean. In this case, specify the fully qualified object name of an MBean in the `javax.management.ObjectName` format. For example: `"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 stream named `helloworldOutstream` of the `helloworld` application:

```
prompt> java wlevs.Admin
        -url service:jmx:rmi:///jndi/rmi://ariel:1099/jmxrmi
        -username wlevs -password wlevs
        SET -mbean
com.bea.wlevs:Name=helloworldOutstream,Type=Stream,Application=helloworld
        -property  MaxSize 1024
```

Note: For clarity, the preceding example is shown on multiple lines; however, when you run the command, enter all arguments and commands on a single line.

For more information about the environment in which this example runs, see "Example Environment" on page 5-5.

CHAPTER 6

Configuring Security for WebLogic Event Server

This section contains information on the following subjects:

Overview of Security in WebLogic Event Server

WebLogic Event Server security can be configured to use one of several types of security providers. As initially installed, it is configured to use the file-based providers for authentication and authorization, with an administrator user with username `wlevs` and password `wlevs`. You can add your own users to this configuration, or you can configure the system to authenticate users from an LDAP or DBMS provider. The file-based authorization provider gives you a pre-configured mapping of roles to permissions. You can use one of the other authorization providers to create your own mappings instead.

Configuring Security With the File-Based Provider

The type of file-based provider security you can enable for WebLogic Event Server is basic authentication. Authentication is a process whereby the identity of users is proved or verified. Authentication typically involves username/password combinations.

All WebLogic Event Server examples and domains are configured to have an administrator with username `wlevs` and password `wlevs`.

By default, security is disabled in the HelloWorld example and in the template user domain located in the *`BEA_HOME`*`/user_projects/domains/wlevs20_domain` directory. 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 WebLogic Event Server 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.

The following table describes the available WebLogic Event Server security roles, as well as the name of the groups that are assigned to these roles.

Table 6-1 Available WebLogic Event Server Roles and Groups

Role	Description	Associated Group Name
Admin	WebLogic Event Server Administrator. Can perform all WebLogic Event Server tasks: start a server instance, deploy applications, and execute all commands of the `wlevs.Admin` utility. By default, all domains and examples include an administrator called `wlevs` with password `wlevs`.	`wlevsAdministrators`
Monitor	Can perform only monitoring tasks, such as the read-only commands of the `wlevs.Admin` utility.	`wlevsMonitors`

Configuring File-Based Security: Main Steps

The following procedure describes the steps to enable file-based provider security for your Weblogic Event Server domain. The procedure also shows how to add two users, one in the Admin role and the other in the Monitor role; this step is not required if the pre-configured `wlevs` username (with password `wlevs`) is adequate. For clarity, it is assumed that:

- The new user in the Admin role is called `jane` with password `secret`.
- The new user in the Monitor role is called `bob` with password `supersecret`.

To configure security:

1. Open a command window and set your environment as described in Setting Up Your Development Environment.

2. Add the *`WLEVS_HOME`*`/bin` directory to your `PATH` environment variable, where *`WLEVS_HOME`* is the main WebLogic Event Server installation directory, such as `d:\beahome\wlevs20`:

   ```
   prompt> set PATH=d:\beahome\wlevs20\bin;%PATH%  (Windows)
   ```

   ```
   prompt> PATH=/beahome/wlevs20/bin:$PATH (UNIX)
   ```

3. Change to the *`DOMAIN_DIR`*`/config` directory, where *`DOMAIN_DIR`* refers to the main directory of your domain, such as `d:\beahome\user_projects\wlevs20_domain`

   ```
   prompt> cd d:\beahome\user_projects\domains\wlevs20_domain
   ```

4. Execute the `secgen.cmd` (Windows) or `secgen.sh` (UNIX) script to generate a unique key for your installation:

   ```
   prompt> secgen -k -o security-key.dat
   ```

   See "The secgen Command Line Utility" on page 6-10 for additional information.

5. Create a new security configuration file that uses the new key by executing the following secgen command:

   ```
   prompt> secgen -F -o security.xml
   ```

   See "The secgen Command Line Utility" on page 6-10 for additional information.

6. Execute the `passgen.cmd` (Windows) or `passgen.sh` (UNIX) script to generate encrypted passwords for the new users; pass as the single argument the cleartext password. Use the output of the script to copy and paste the encrypted password.

   ```
   prompt> passgen.cmd secret
   {SHA-1}C19+kCHEcB1ZQ/ZjuiQxNsNbFGoQZYw=
   ```

   ```
   prompt> passgen.cmd supersecret
   {SHA-1}LoLDn/H1aC2qSoExZ+yNBnl1Tbqys/8=
   ```

   See "The passgen Command Line Utility" on page 6-7 for additional information.

7. Using your favorite text editor, edit the file `atnstore.txt` following these guidelines:

   – For each new user you want to configure, add two lines at the beginning of the file. Start each line with the `user:` and `password:` keywords, then enter the value. Be sure you enter the encrypted password you generated in a preceding step.

You must also add a blank line between each user entry, as shown in the example below.

– To assign a user to the Admin role, add a `member: user` entry below the `group: wlevsAdministrators` entry, where *user* refers to the username.

– To assign a user to the Monitor role, add a `member: user` entry below the `group: wlevsMonitors` entry, where *user* refers to the username.

The following sample `atnstore.txt` file shows configurations for the `wlevs`, `jane`, and `bob` users:

```
user: wlevs
password: {SHA-1}8MHYCQlzyh/S7TAbEC+fU34o4rf7GVM=

user: jane
password: {SHA-1}C19+kCHEcB1ZQ/ZjuiQxNsNbFGoQZYw=

user: bob
password: {SHA-1}LoLDn/H1aC2qSoExZ+yNBnl1Tbqys/8=

group: wlevsAdministrators
member: wlevs
member: jane

group: wlevsMonitors
member: bob
```

8. Update the `startwlevs.cmd` (Windows) or `startwlevs.sh` (UNIX) start script by adding the following two arguments to the `java` command that starts WebLogic Event Server:

```
-Dcom.bea.core.security.username=username
-Dcom.bea.core.security.password=password
```

where *username* and *password* refer to the administrator user you have configured for WebLogic Event Server, `wlevs` by default. The server start scripts are located in the main directory of your domain, such as *BEA_HOME*`/user_projects/domains/wlevs20_domain`.

The following snippet from the Windows `startwlevs.cmd` script shows in bold how to specify the administrator username and password:

```
 %JAVA_HOME%\bin\java -Dwlevs.home=%USER_INSTALL_DIR%
-Dbea.home=%BEA_HOME% -Dcom.bea.core.security.username=jane
-Dcom.bea.core.security.password=secret -jar
"..\..\..\bin\wlevs_2.0.jar" %1 %2 %3 %4 %5 %6
```

If you do not want to put cleartext passwords in the `startwlevs.cmd` script, see "Avoiding Cleartext Passwords in the startwlevs Script" on page 6-5.

9. Restart WebLogic Event Server for the security to take effect. See "Stopping and Starting the Server" on page 2-2.

After you have enabled security, you must use the `-username` and `-password` arguments to the Deployer and wlevs.Admin tool to specify the administrator user.

Avoiding Cleartext Passwords in the startwlevs Script

The procedure in "Configuring File-Based Security: Main Steps" on page 6-2 describes how to update the `startwlevs` server start script with the cleartext password of the administrator user. Cleartext passwords, however, are typically not allowed in a production environment. There are two ways to not use cleartext passwords when starting the server:

- "Being Prompted for a Password at Server Startup" on page 6-5
- "Putting the Encrypted Password in a Security XML file" on page 6-5

Being Prompted for a Password at Server Startup

If you want the server start script to prompt you for the administrator password, then update the script by adding the following argument to the `java` command that starts WebLogic Event Server:

```
-Dcom.bea.core.security.prompt=true
```

In this case do *not* specify the `com.bea.core.security.username` or `com.bea.core.security.password` arguments in the script.

When the script prompts you for a password, the password field will not be echoed to the screen on platforms where this is supported.

The following snippet from the Windows `startwlevs.cmd` script shows in bold how to specify that you be prompted for a password when starting the server:

```
 %JAVA_HOME%\bin\java -Dwlevs.home=%USER_INSTALL_DIR%
-Dbea.home=%BEA_HOME% -Dcom.bea.core.security.prompt=true -jar
"..\..\..\bin\wlevs_2.0.jar" %1 %2 %3 %4 %5 %6
```

Putting the Encrypted Password in a Security XML file

Follow these steps if you want to put the encrypted administrator password in an XML file that will then be accessed by the server start script.

1. Open a command window and set your environment as described in Setting Up Your Development Environment.

2. Change to the *DOMAIN_DIR*/config directory, where *DOMAIN_DIR* refers to the main directory of your domain, such as d:\beahome\user_projects\wlevs20_domain
3. Using your favorite XML editor, edit the security-config.xml file by adding an <msa-security> child element to the <config> root element. Use the two child elements of <msa-security> shown below to specify the administrator's username and cleartext password:

   ```
   <?xml version="1.0" encoding="UTF-8"?>
   <config>
     <css-realm>
     ...
     <msa-security>
       <boot-user-name-encrypted>jane</boot-user-name-encrypted>
       <password>secret</password>
     </msa-security>
   </config>
   ```

4. Execute the following java command to encrypt the cleartext password in the security-config.xml file:

   ```
   prompt> java -jar BEA_HOME/modules/com.bea.core.bootbundle_3.0.1.0.jar .
   security-config.xml
   ```

   where *BEA_HOME* refers to the main BEA directory into which you installed WebLogic Event Server, such as d:\beahome. The second argument refers to the directory that contains the security-config.xml file; because this procedure directs you to change to the directory, the example shows ".".

   After you run the command, the security-config.xml file will use encrypted passwords, such as the following:

   ```
   <?xml version="1.0" encoding="UTF-8"?>
   <config>
     <css-realm>
     ...
     <msa-security>

   <boot-user-name-encrypted>{Salted-3DES}dBcBdaoemdY=</boot-user-name-enc
   rypted>

       <password>{Salted-3DES}X86x/+El2/s=</password>
     </msa-security>
   </config>
   ```

5. Copy the file .msaInternal.dat, generated by the java command in the preceding step, from the directory in which you ran the java command to the main domain directory. For example:

```
prompt> copy
d:\beahome\user_projects\wlevs20_domain\config\.msaInternal.dat
d:\beahome\user_projects\wlevs20_domain
```

Disabling File-Based Security

To disable file-based security in a domain, add the `-disablesecurity` flag to the `java` command that starts WebLogic Event Server in the `startwlevs.cmd` (Windows) or `startwlevs.sh` (UNIX) start script. This script is located in the main directory of your domain, such as *`BEA_HOME`*`/user_projects/domains/wlevs20_domain`.

The following snippet from the Windows `startwlevs.cmd` script shows in bold how to disable security:

```
%JAVA_HOME%\bin\java -Dwlevs.home=%USER_INSTALL_DIR% -Dbea.home=%BEA_HOME% -jar
"%USER_INSTALL_DIR%\bin\wlevs_2.0.jar" -disablesecurity %1 %2 %3 %4 %5 %6
```

The passgen Command Line Utility

Use the `passgen` command line utility to hash user passwords for addition to a security database.

The `passgen` utility is located in the *`WLEVS_HOME`*`/bin` directory, where *`WLEVS_HOME`* is the main WebLogic Event Server installation directory, such as `d:\beahome\wlevs20`. The utility comes in two flavors:

- `passgen.cmd` (Windows)
- `passgen.sh` (UNIX)

passgen Syntax

```
prompt> passgen [-a algorithm] [-s saltsize] [-h] [-?] [password]*
```

where:

Option	Description	Default Value
`-a`	*`algorithm`* specifies the hash algorithm to use: • SHA-1 • MD2 • MD5 • SSHA • SHA-256 **Note:** 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 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:

```
$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:

- "Using passgen interactively" on page 6-9
- "Providing a Password on the Command Line" on page 6-9

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 generates a security key or a security configuration file that uses encrypted passwords.

The `secgen` utility is located in the *`WLEVS_HOME/`*`bin` directory, where *`WLEVS_HOME`* is the main WebLogic Event Server installation directory, such as `d:\beahome\wlevs20`. 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.

```
  prompt> secgen -F [-o outputfile] [-i inputkeyfile] [-e] [-P
PropertyFilePath]
```

where:

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.

Option	Description	Comments
`-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 "Using the secgen Properties File" on page 6-11 for details.	A `SecGenTemplate.properties` template file is located at *`WLEVS_HOME`*`/bin` where *`WLEVS_HOME`* is the main installation directory of WebLogic Event Server, such as `/beahome/wlevs20`.

Generating a Key File

Use the following command line options to generate a security key file.

```
prompt> secgen [-k] [-o outputfile]
```

where:

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 *`WLEVS_HOME`*`/bin` where *`WLEVS_HOME`* is the main installation directory of WebLogic Event Server, such as `/beahome/wlevs20`.

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.

CHAPTER 7

Configuring Jetty for WebLogic Event Server

This section contains information on the following subjects:

- "Overview Of Jetty Support in WebLogic Event Server" on page 7-1
- "Configuring a Jetty Server Instance" on page 7-4
- "Example Jetty Configuration" on page 7-8

Overview Of Jetty Support in WebLogic Event Server

WebLogic Event Server supports Jetty as Java Web server to deploy HTTP servlets and static resources.

WebLogic Event Server support for Jetty is based on Version 1.2 the OSGi HTTP Service. This API provides ability to dynamically register and unregister `javax.servlet.Servlet` objects with the runtime and static resources. This specification requires at minimum version 2.1 of the Java Servlet API.

WebLogic Event Server supports the following features for Jetty:

- "Servlets" on page 7-2
- "Network I/O Integration" on page 7-2
- "Thread Pool Integration" on page 7-2
- "Work Managers" on page 7-2

For details about configuring Jetty, see "Configuring a Jetty Server Instance" on page 7-4.

Servlets

In addition to supporting typical (synchronous) Java servlets, WebLogic Event Server 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

WebLogic Event Server 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

WebLogic Event Server Jetty services use the WebLogic Event Server Work Manager to provide for scalable thread pooling. See "</config>" on page 7-9.

Note: Jetty provides its own thread pooling capability. However, BEA recommends using the WebLogic Event Server self-tuning thread pool to minimize footprint and configuration complexity.

Work Managers

WebLogic Event Server allows you to configure how your application prioritizes the execution of its work. Based on rules you define and by monitoring actual runtime 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.

Understanding How WebLogic Event Server Uses Thread Pools

WebLogic Event Server uses is a single thread pool, in which all types of work are executed. WebLogic Event Server 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, WebLogic Event Server increases the thread count. Similarly, if statistics indicate that fewer threads did not reduce throughput, WebLogic Event Server decreases the thread count.

Understanding Work Manager

WebLogic Event Server 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 application. At run-time, WebLogic Event Server 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 WebLogic Event Server 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, WebLogic Event Server 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 WebLogic Event Server does not schedule requests of this type until the number of concurrent executions falls below the limit. The WebLogic Event Server 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 WebLogic Event Server instance is close to a deadlock condition. In that case, it the constraint causes WebLogic Event Server 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 WebLogic Event Server domain:

- `<jetty>`: See "jetty Configuration Object" on page 7-4 for details.
- `<netio>`: See "netio Configuration Object" on page 7-5 for details.
- `<work-manager>`: See "work-manager Configuration Object" on page 7-6 for details.

Use the `<jetty-web-app>` configuration object to define a Web application in the Jetty instance; see "jetty-web-app Configuration Object" on page 7-6 for details.

See "Example Jetty Configuration" on page 7-8 for a sample of using each of the preceding configuration objects.

jetty Configuration Object

Use the parameters described in the following table to define a `<jetty>` configuration object in your `config.xml` file.

Table 7-1 Configuration Parameters for <jetty>

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 "netio Configuration Object" on page 7-5 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 "work-manager Configuration Object" on page 7-6.
`scratch-directory`	`String`	The name of a directory where temporary files required for web apps, 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 7-2 Configuration Parameters for <netio>

Parameter	Type	Description
`name`	`String`	The name of this configuration object.

Table 7-2 Configuration Parameters for <netio>

Parameter	Type	Description
`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. **Note:** 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 7-3 Configuration Parameters for <work-manager>

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 7-4 Configuration Parameters for <jetty-web-app>

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 "Configuring a Jetty Server Instance" on page 7-4. If not specified, a directory is created at????.
`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 "Configuring a Jetty Server Instance" on page 7-4.
`name`	`String`	The name of this configuration object.

Developing Servlets for Jetty

WebLogic Event Server supports development of servlets for deployment to Jetty by creating a standard J2EE Web Application and configuring it using the "jetty-web-app Configuration Object" on page 7-6.

Web App Deployment

WebLogic Event Server 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 apps 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:

Listing 7-1 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>
```

CHAPTER 8

Configuring JMX for WebLogic Event Server

This section contains information on the following subjects:

Overview of JMX Support in WebLogic Event Server

WebLogic Event Server provides standards-based interfaces that are fully compliant with the Java Management Extensions (JMX) specification. Software vendors can use these interfaces to monitor WebLogic Server MBeans, to change the configuration of a WebLogic Server domain, and to and monitor the distribution (activation) of those changes to all server instances in the domain.

The `wlevs.Admin` utility uses JMX to connect to a server so you can manipulate its MBean instances, in particular to view, add, and update the EPL rules associated with the processors of a particular WebLogic Event Server application.

However, to use the `wlevs.Admin` utility, and the JMX interfaces in general, you must configure WebLogic Event Server with the JMX configuration information in the `config.xml` file.

Configuring JMX

You use the following configuration objects to configure an instance of the Jetty HTTP server in the `config.xml` file that describes your WebLogic Event Server domain:

- `<jmx>`: See "jmx Configuration Object" on page 8-2 for details.
- `<rmi>`: See "rmi Configuration Object" on page 8-2 for details.
- `<jndi-context>`: See "jndi-context Configuration Object" on page 8-3 for details.
- `<exported-jndi-context>`: See "exported-jndi-context Configuration Object" on page 8-4 for details

See "Example of Configuring JMX" on page 8-5 for a sample of using each of the preceding configuration objects.

jmx Configuration Object

The following table describes the configuration information for the `<jmx>` element in the `config.xml` file.

Table 8-1 Configuration Parameters for <jmx>

Parameter	Type	Description
`rmi-service-name`	`String`	The name of the RMI service with which the jmx server will register to receive calls.
`rmi-jrmp-port`	`int`	The port on which to listen for RMI JRMP JMX requests
`jndi-service-name`	`String`	The name of the JNDI service to which the jmx server will bind its object.
`rmi-registry-port`	`int`	The port on which to start the RMIRegistry

rmi Configuration Object

The WebLogic Event Server 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.

The following table shows the parameters of the `<rmi>` configuration object that you use to export server-side objects to remote clients.

Table 8-2 Configuration Parameters for <rmi>

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 "Overview Of Jetty Support in WebLogic Event Server" on page 7-1).
`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 WebLogic Event Server, that initializes the JNDI environment to support the factory manager bundle.

Use the following configuration object to configure the `<jndi-context>` configuration object.

Table 8-3 Configuration Parameters for <jndi-context>

Parameter	Type	Description
`default-provider`	`boolean`	If `true`, the default WebLogic Event Server JNDI provider is used. Default value is `true`.
`name`	`String`	The name of this configuration object.

exported-jndi-context Configuration Object

Note: Requires a configured "jndi-context Configuration Object" on page 8-3.

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 8-4 Configuration Parameters for <exported-jndi-context>

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 "rmi Configuration Object" on page 8-2.
`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

The following `config.xml` snippet shows an example of configuring JMX; only relevant parts of the file are shown.

Listing 8-1

```
<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>
      <rmi-registry-port>10099</rmi-registry-port>
      <rmi-jrmp-port>9999</rmi-jrmp-port>
    </jmx>

  </config>
```

CHAPTER 9

Configuring Access to a Relational Database

This section contains information on the following subjects:

- "Overview of Database Access from a WebLogic Event Server Application" on page 9-1
- "Configuring Access to a Relational Database: Main Steps" on page 9-4

Overview of Database Access from a WebLogic Event Server Application

WebLogic Event Server supports Java Database Connectivity (JDBC) 3.0 for relational database access.

The JDBC API 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.

WebLogic Event Server provides the following JDBC drivers, installed in the *`WLEVS_HOME`*`/bin` directory, where *`WLEVS_HOME`* refers to the main installation directory such as `d:\beahome\wlevs20`.

- Oracle 10.2.0 thin driver (packaged in the `com.bea.oracle.ojdbc14_10.2.0.jar` JAR file)

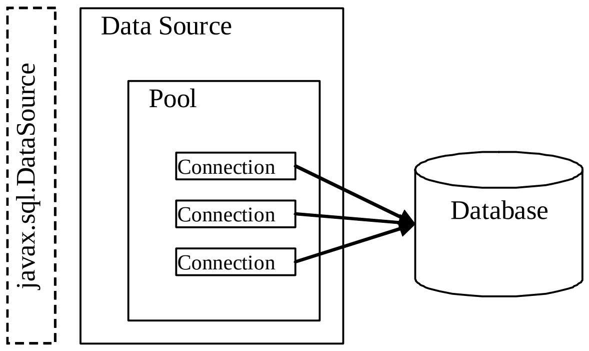
- Microsoft SQL (packaged in the `com.bea.microsoft.sqljdbc_1.0.jar` JAR file)

WebLogic Event Server also provides a `DataSource` abstraction that encapsulates a JDBC driver `DataSource` object and manages a pool of pre-established connections.

Description of WebLogic Event Server Data Sources

WebLogic Event Server `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 9-1 Data Source



You are required to configure a WebLogic Event Server `DataSource` in the server's `config.xml` file if you want to access a relational database from an EPL `rule`; for details, see Configuring the Complex Event Processor. 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.

Data Source Configuration

The WebLogic Event 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. Following is a sample datasource `config.xml` section that illustrates the configuration format.

```
<data-source>
  <name>oraxads2</name>
```

```
    <driver-params>
      <url>jdbc:oracle:thin:@buckhorn.bea.com:1521:ce102a</url>
      <driver-name>oracle.jdbc.xa.client.OracleXADataSource</driver-name>
      <properties>
        <element>
          <name>user</name>
          <value>cedeployqa</value>
        </element>
        <element>
          <name>password</name>
          <value>cedeployqa123</value>
        </element>
      </properties>
      <use-xa-data-source-interface>true</use-xa-data-source-interface>
    </driver-params>
    <connection-pool-params>
      <initial-capacity>15</initial-capacity>
      <max-capacity>50</max-capacity>
      <test-table-name>SQL SELECT 1 FROM DUAL</test-table-name>
    </connection-pool-params>
    <data-source-params>
      <jndi-names>
        <element>oraxads2</element>
      </jndi-names>

<global-transactions-protocol>TwoPhaseCommit</global-transactions-protocol
>
    </data-source-params>
  </data-source>

  <transaction-manager>
    <name>tm1</name>
    <rmi-service-name>RMI</rmi-service-name>
  </transaction-manager>
```

A data source depends on the availability of a local transaction manager, which you configure using the `<transaction-manager>` element of `config.xml` as shown above. The transaction manager in turn depends on a configured RMI object, as described in "rmi Configuration Object" on page 8-2.

Configuring Access to a Relational Database: Main Steps

Follow these steps to configure and use JDBC in your application:

1. Update the server start script in your domain directory so that WebLogic Event Server 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 main domain directory. The out-of-the-box sample domains are located in `WLEVS_HOME/samples/domains`, and the user domains are located in `BEA_HOME/user_projects/domains`, where `WLEVS_HOME` refers to the main WebLogic Event Server installation directory, such as `d:\beahome\wlevs20`, and `BEA_HOME` refers to the directory above `WLEVS_HOME`, such as `d:\beahome`.

   Update the start script by adding the `-Xbootclasspath/a` option to the Java command that executes the `wlevs_2.0.jar` file. Set the `-Xbootclasspath/a` option to the full pathname of the 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 (updated section shown in bold):

   ```
     %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_2.0.jar" -disablesecurity %1 %2
   %3 %4 %5 %6
   ```

   In the example, `%USER_INSTALL_DIR%` points to `WLEVS_HOME`.

2. If are configuring JDBC for use in an EPL rule, or you want to use the WebLogic Event Server `DataSource`, configure the data source in the server's `config.xml` file.

   For details, see "Data Source Configuration" on page 9-2.

3. If WebLogic Event Server is running, restart it so it reads the new `java` option. See Stopping and Starting the Server.

CHAPTER 10

Configuring Logging and Debugging

This section contains information on the following subjects:

Configuration Scenarios

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.

Overview of Logging Services Configuration

This release 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`.

- "Setting the Log Factory" on page 10-2
- "Using Log Severity Levels" on page 10-3
- "Log Message Format" on page 10-4
- "OSGI Framework Logger" on page 10-5

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.LogFa
  ctoryImpl
  ```

- You can programmatically implement the logging by:

  ```
  import org.apache.commons.logging.LogFactory;
  
  System.setProperty(LogFactory.FACTORY_PROPERTY, "weblogic.logging.commo
  ns.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.log
ging.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`, which 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 to highest impact:

`TRACE`, `DEBUG`, `INFO`, `NOTICE`, `WARNING`, `ERROR`, `CRITICAL`, `ALERT`, `EMERGENCY`

You can set a log severity level on the logger and the handler. 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.

Table 10-1 lists the severity levels of log messages.

Table 10-1 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.
`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.

Table 10-1 Message Severity (Continued)

Severity	Meaning
`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 Message Format

The system writes a message to sdtout and the specified log file, consisting of the Timestamp, Severity, Subsystem, and the Message, along with the stacktrace if any. Each attribute is contained between angle brackets.

The following is an example of a message in the server log file:

```
<May 02, 2007 10:46:51 AM EST> <Notice> <CommonTestSubsystem> <BEA-123456>
<Another Commons test message>
```

Format of Output to Standard Out and Standard Error

When the system writes a message to standard out, the output does not include the #### prefix and does not include the Server Name, Machine Name, Thread ID, User ID, Transaction ID, Diagnostic Context ID, and Raw Time Value fields.

The following is an example of how the message from the previous section would be printed to standard out:

```
<Sept 22, 2004 10:51:10 AM EST> <Notice> <WebLogicServer> <BEA-000360>
<Server started in RUNNING mode>
```

In this example, the message attributes are: Locale-formatted Timestamp, Severity, Subsystem, Message ID, and Message Text.

OSGI Framework Logger

WebLogic Event Server 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 equals 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.

How to Use the Commons Logging API

To use Commons Logging:

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.

Listing 10-1 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 the WebLogic Event Server Logging Service

The following sections provide information on configuring WebLogic Event Server logging:

- "Logging Service" on page 10-6
- "log-stdout" on page 10-7
- "log-file" on page 10-8

Logging Service

This section provides information on the logging-service configuration object:

Table 10-2 Configuration Parameters for logging-service

Parameter	Type	Description
`log-file-config`	`String`	The configuration of the log file and its rotation policies. See "log-file" on page 10-8.
`stdout-config`	`String`	The name of the `stdout` configuration object used to configure `stdout` output. See "log-stdout" on page 10-7.
`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`, `Trace`
`name`	`String`	The name of this configuration object.

log-stdout

This section provides information on the `log-stdout` configuration object:

Table 10-3 Configuration Parameters for log-stdout

Parameter	Type	Description
`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`.
`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.`
`name`	`String`	The name of this configuration object.

log-file

This section provides information on the `log-file` configuration object:

Table 10-4 Configuration Parameters for log-file

Parameter	Type	Description
`number-of-files-limited`	`Boolean`	If `true`, old rotated files are deleted. Default is `false`.
`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`.
`base-log-file-name`	`String`	The log file name. Default value is `server.log`.
`rotate-log-on-startup-enabled`	`Boolean`	If `true`, the log file is rotated on startup. Default value is `true`.

Table 10-4 Configuration Parameters for log-file

Parameter	Type	Description
`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.
`name`	`String`	The name of this configuration object.

Debug

The following sections provide information on how to use the WebLogic Event Server debugging feature:

- "Configuring debug using System Properties" on page 10-10
- "Configuring debug using a Configuration File" on page 10-11
- "Supported Debug Flags" on page 10-11
- "Example Debug Configuration" on page 10-14

Configuring debug using System Properties

You can set a system property on the Java command line by using the following steps:

1. Create a property by prepending `-D` to the flag
2. Turn the flag on by setting the property to `true`.

For example: `-Dcom.bea.core.debug.DebugSDS=true`

Configuring debug using a Configuration File

Use the following steps to configure debugging from a configuration file:

1. Create an XML tag by dropping "`com.bea.core.debug.`" from the flag name.
2. Turn the flag on by setting the flag to `true`.
3. Wrap with `debug-properties` tag.
4. Set `logger-severity` to `Debug` in the logging service stanza.
5. Set `stdout-severity` to `Debug` in the stdout configuration stanza.

See Example Debug Configuration.

Supported Debug Flags

The following table provides the supported debug flags for this release:

Table 10-5 Debug Flags

Debug Flag	Description
`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.servicehelper.dumpstack`	Dump stack traces when servicehelper times 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.DebugCM`	Configuration Manager
`com.bea.core.debug.DebugCM.stdout`	Configuration Manager debug strings go to stdout
`com.bea.core.debug.DebugCSSServices`	CSS Services

Table 10-5 Debug Flags

Debug Flag	Description
`com.bea.core.debug.DebugCSSServices.stdout`	CSS Services debug strings go to stdout
`com.bea.core.debug.DebugCSS`	CSS
`com.bea.core.debug.DebugCSS.stdout`	CSS debug strings go to stdout
`com.bea.core.debug.DebugBootBundle`	Boot Debugging
`com.bea.core.debug.DebugBootBundle.stdout`	Boot Debugging debug strings go to stdout
`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.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

Table 10-5 Debug Flags

Debug Flag	Description
`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.DebugStoreAdmin`	Store Administration
`com.bea.core.debug.DebugStoreIOPhysical`	Store IOPhysical
`com.bea.core.debug.DebugStoreIOPhysicalVerbose`	Store IOPhysicalVerbose
`com.bea.core.debug.DebugStoreIOLogical`	Store IOLogical
`com.bea.core.debug.DebugStoreIOLogicalBoot`	Store IOLogicalBoot
`com.bea.core.debug.DebugStoreXA`	Store XA
`com.bea.core.debug.DebugStoreXAVerbose`	Store XAVerbose
`com.bea.core.debug.DebugConfigurationRuntime`	Runtime informatin from the Runtime MBeans
`com.bea.core.debug.DebugJDBCInternal`	JDBC Internal
`com.bea.core.debug.DebugJTAJDBC`	JTA JDBC
`com.bea.core.debug.DebugJDBCSQL`	JDBC SQL
`com.bea.core.debug.DebugJDBCRMI`	JDBC RMI
`com.bea.core.debug.DebugJDBCConn`	JDBC Connection

Table 10-5 Debug Flags

Debug Flag	Description
`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 stdout

Example Debug Configuration

The following code provides an example debug configuration to turn on Simple Declarative Services (SDS) debugging in the `config.xml file`:

Listing 10-2 Example debug Configuration

```
<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>
```

Log4j

The following sections provide information on using Log4j:

- "About Log4j" on page 10-15
- "log4j Properties" on page 10-16
- "Enabling Log4j Logging" on page 10-16

About Log4j

Log4j is an open source tool developed for putting log statements in your application. Log4j has three main components:

- "Loggers" on page 10-15
- "Appenders" on page 10-16
- "Layouts" on page 10-16

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/docs/`
- `http://logging.apache.org/log4j/docs/api/index.html`
- Short introduction to log4j at `http://logging.apache.org/log4j/docs/manual`.

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.

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:

Listing 10-3 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
```

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
.LogFactoryimpl

   -Dorg.apache.commons.logging.Log=org.apache.commons.logging.impl.Log4JL
ogger

   -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.