Oracle® Fusion Middleware

Configuring Log Files and Filtering Log Messages for Oracle WebLogic Server

14c (14.1.2.0.0)
F62099-01
December 2024

Oracle Fusion Middleware Configuring Log Files and Filtering Log Messages for Oracle WebLogic Server, 14c (14.1.2.0.0)

F62099-01

Primary Author: Oracle Corporation

Contents

Preface

Oracle WebLogic Server logging services is used to monitor server, subsystem, and application events. You can configure WebLogic Server to write messages to log files and listen for the log messages that WebLogic Server broadcasts. You can also view log messages through the WebLogic Remote Console.

Audience

This document is a resource for system administrators who configure WebLogic logging services and monitor server and subsystem events, and for Java Platform, Enterprise Edition (Jakartaa EE) application developers who want to integrate their application logs with WebLogic Server logs. This document is relevant to all phases of a software project, from development through test and production phases.

This document does not address application logging or localization and internationalization of log message catalogs. For links to information on these topics, see Related Documentation.

It is assumed that the reader is familiar with Jakartaa EE and Web technologies, object-oriented programming techniques, and the Java programming language.

Documentation Accessibility

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

Access to Oracle Support

Oracle customer access to and use of Oracle support services will be pursuant to the terms and conditions specified in their Oracle order for the applicable services.

Diversity and Inclusion

Oracle is fully committed to diversity and inclusion. Oracle respects and values having a diverse workforce that increases thought leadership and innovation. As part of our initiative to build a more inclusive culture that positively impacts our employees, customers, and partners, we are working to remove insensitive terms from our products and documentation. We are also mindful of the necessity to maintain compatibility with our customers' existing technologies and the need to ensure continuity of service as Oracle's offerings and industry standards evolve.

Because of these technical constraints, our effort to remove insensitive terms is ongoing and will take time and external cooperation.

Related Documentation

The corporate Web site provides all documentation for WebLogic Server. Specifically, View Logs and Configure Logs in the *Oracle WebLogic Remote Console Online Help* describes configuring log files and log messages that a WebLogic Server instance generates.

Using Message Catalogs with WebLogic Server in *Adding WebLogic Logging Services to Applications Deployed on Oracle WebLogic Server* describes how you can use WebLogic Server message catalogs, non-catalog logging, and servlet logging to produce log messages from your application or a remote Java client, and describes WebLogic's support for internationalization and localization of log messages.

- Logging Samples and Tutorials
- New and Changed WebLogic Server Features

Logging Samples and Tutorials

Oracle provides a variety of logging code examples and tutorials that show WebLogic Server logging configuration and API use, and provide practical instructions on how to perform key development tasks. For more information, see Sample Applications and Code ExamplesSample Applications and Code Examples in Understanding Oracle WebLogic Server.

Logging Examples in the WebLogic Server Distribution

WebLogic Server optionally installs API code examples in *ORACLE_HOME*`\wlserver\samples\server`, where *ORACLE_HOME* represents the directory in which you installed WebLogic Server.

New and Changed WebLogic Server Features

For a comprehensive listing of the new WebLogic Server features introduced in this release, see *What's New in Oracle WebLogic Server*.

Conventions

The following text conventions are used in this document:

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

1

Understanding WebLogic Logging Services

WebLogic logging services provide facilities for writing, viewing, filtering, and listening for log messages. These log messages are generated by WebLogic Server instances, subsystems, and Jakarta EE applications that run on Oracle WebLogic Server or in client JVMs.

- What You Can Do With WebLogic Logging Services
- How WebLogic Logging Services Work
  Learn about the WebLogic Server logging environment and the logging process.
- Server and Subsystem Logs
  Each subsystem within WebLogic Server generates log messages to communicate its status.
- Log Message Format
  When a WebLogic Server instance writes a message to the server log file, the first line of each message begins with `####` followed by the message attributes. Each attribute is contained between angle brackets.
- Message Attributes
- Message Severity
- Viewing WebLogic Server Logs
  The WebLogic Remote Console provides a log viewer for all the log files in a domain.
- Configuring java.util.logging Logger Levels
  WebLogic Server supports configuring `java.util.logging.Logger` levels for named loggers in the JDK `LogManager` from within the WebLogic Server logging configuration.

What You Can Do With WebLogic Logging Services

WebLogic Server subsystems use logging services to provide information about events such as the deployment of new applications or the failure of one or more subsystems. A server instance uses them to communicate its status and respond to specific events. For example, you can use WebLogic logging services to report error conditions or listen for log messages from a specific subsystem.
Each WebLogic Server instance maintains a server log. Because each WebLogic Server domain can run concurrent, multiple instances of WebLogic Server, the logging services collect messages that are generated on multiple server instances into a single, domain-wide message log. The domain log provides the overall status of the domain. See Server Log Files and Domain Log Files.

How WebLogic Logging Services Work

Learn about the WebLogic Server logging environment and the logging process.

- Components and Environment
- Terminology
- Overview of the Logging Process

- Server Log Files and Domain Log Files
- How a Server Instance Forwards Messages to the Domain Log

Components and Environment

There are two basic components in any logging system: a component that produces log messages and another component to distribute (publish) messages. WebLogic Server subsystems use a message catalog feature to produce messages and the Java Logging APIs to distribute them, by default. Developers can also use message catalogs for applications they develop.

The message catalog framework provides a set of utilities and APIs that your application can use to send its own set of messages to the WebLogic server log. The framework is ideal for applications that need to localize the language in their log messages, but even for those applications that do not need to localize, it provides a rich, flexible set of tools for communicating status and output.

See Using Message Catalogs with WebLogic Server in *Adding WebLogic Logging Services to Applications Deployed on Oracle WebLogic Server*.

In addition to using the message catalog framework, your application can use the following mechanisms to send messages to the WebLogic server log:

- `weblogic.logging.NonCatalogLogger` APIs

  With `NonCatalogLogger`, instead of calling messages from a catalog, you place the message text directly in your application code. See Using the NonCatalogLogger APIs in *Adding WebLogic Logging Services to Applications Deployed on Oracle WebLogic Server*.

- Server Logging Bridge

  WebLogic Server provides a mechanism by which your logging application can have its messages redirected to WebLogic logging services without the need to make code changes or implement any of the propriety WebLogic Logging APIs.

Use of either the `NonCatalogLogger` APIs or Server Logging Bridge is suitable for logging messages that do not need to be internationalized or that are internationalized outside the WebLogic I18n framework.

To distribute messages, WebLogic Server supports Java based logging by default. The `LoggingHelper` class provides access to the `java.util.logging.Logger` object used for server logging. This lets developers take advantage of the Java Logging APIs to add custom handlers, filters, and formatters. See the `java.util.logging` API documentation at `http://docs.oracle.com/javase/8/docs/api/java/util/logging/package-summary.html`.

Terminology

To understand the WebLogic Logging services, you must understand the following terminology associated with it:

- Logger - A `Logger` object logs messages for a specific subsystem or application component. WebLogic logging services use a single instance of `java.util.logging.Logger` for logging messages from the Message Catalogs, `NonCatalogLogger`, and the Debugging system.
- Handler - A class that extends `java.util.logging.Handler` and receives log requests sent to a logger. Each `Logger` instance can be associated with a number of handlers to

which it dispatches log messages. A handler attaches to a specific type of a log message; for example, the File Handler for the server log file.

Overview of the Logging Process

WebLogic Server subsystems or application code send log requests to `Logger` objects. These `Logger` objects allocate `LogRecord` objects which are passed to `Handler` objects for publication. Both loggers and handlers use severity levels and (optionally) filters to determine if they are interested in a particular `LogRecord` object. When it is necessary to publish a `LogRecord` object externally, a handler can (optionally) use a formatter to localize and format the log message before publishing it to an I/O stream.

Figure 1-1 shows the WebLogic Server logging process: WebLogic Catalog APIs or Commons Logging APIs are used for producing messages; Java Logging (default) is the only options for distributing messages.

Figure 1-1 WebLogic Server Logging Process

process

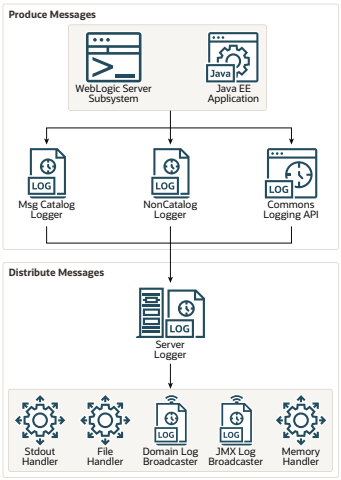


Figure 1-1 illustrates the following process:

1. The client, in this case, a WebLogic Server subsystem or Jakarta EE application, invokes a method on one of the generated Catalog Loggers or the Commons Logging implementation for WebLogic Server.
   a. When WebLogic Server message catalogs and the `NonCatalogLogger` generate messages, they distribute their messages to the server `Logger` object.
   b. The Jakarta Commons Logging APIs define a factory API to get a `Logger` reference which dispatches log requests to the server `Logger` object.

      The server `Logger` object can be an instance of `java.util.logging.Logger`.
2. The server `Logger` object publishes the messages to any message handler that has subscribed to the `Logger`.

   For example, the Stdout Handler prints a formatted message to standard out and the File Handler writes formatted output to the server log file. The Domain Log Broadcaster sends log messages to the domain log, which resides on the Administration Server, and the JMX Log Broadcaster sends log messages to JMX listeners on remote clients.

Server Log Files and Domain Log Files

Each WebLogic Server instance writes all messages from its subsystems and applications to a server log file that is located on the local host computer. By default, the server log file is located in the `logs` directory below the server instance root directory; for example, *DOMAIN_NAME*`\servers\`*SERVER_NAME*`\logs\`*SERVER_NAME*`.log`, where *DOMAIN_NAME* is the name of the directory in which you located the domain and *SERVER_NAME* is the name of the server.

In addition to writing messages to the server log file, each server instance forwards a subset of its messages to a domain-wide log file. By default, servers forward only messages of severity level `Notice` or higher. While you can modify the set of messages that are forwarded, servers can never forward messages of the `Debug` severity level. See Define Debug Settings in *Oracle WebLogic Remote Console Online Help*.

The domain log file provides a central location from which to view the overall status of the domain. The domain log resides in the Administration Server `logs` directory. The default name and location for the domain log file is *DOMAIN_NAME*`\servers\`*ADMIN_SERVER_NAME*`\logs\`*DOMAIN_NAME*`.log`, where *DOMAIN_NAME* is the name of the directory in which you located the domain and *ADMIN_SERVER_NAME* is the name of the Administration Server.

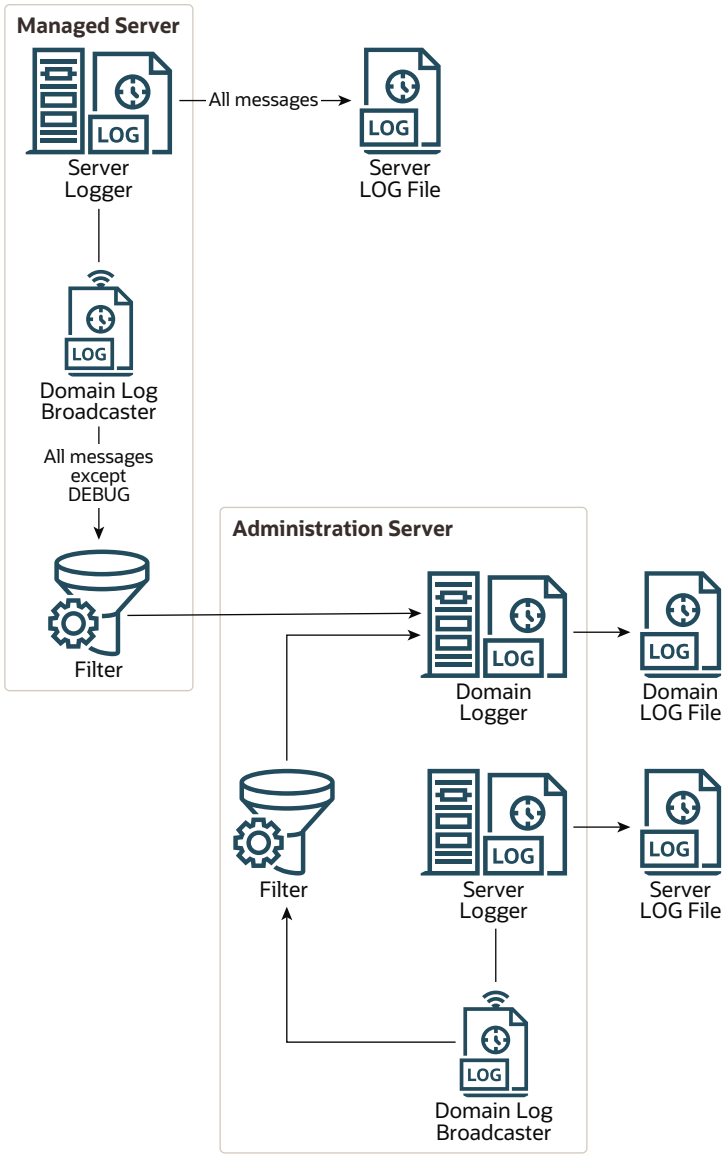
The timestamp for a record in the domain log is the timestamp of the server where the message originated. Log records in the domain log are not written in the order of their timestamps; the messages are written as soon as they arrive. It may happen that a Managed Server remains out of contact with the Administration Server for some period of time. In that case, the messages are buffered locally and sent to the Administration Server once the servers are reconnected.

How a Server Instance Forwards Messages to the Domain Log

To forward messages to the domain log, each server instance broadcasts its log messages. A server broadcasts all messages and message text except for messages of the `Debug` severity level.

The Administration Server listens for a subset of these messages and writes them to the domain log file. To listen for these messages, the Administration Server registers a listener with each Managed Server. By default, the listener includes a filter that allows only messages of severity level `Notice` and higher to be forwarded to the Administration Server. (See Figure 1-2.)

Figure 1-2 WebLogic Server and Domain Logs



For any given WebLogic Server instance, you can override the default filter and create a log filter that causes a different set of messages to be written to the domain log file. For information about setting up a log filter for a WebLogic Server instance, see Create a Log Filter in *Oracle WebLogic Remote Console Online Help*.

If the Administration Server is unavailable, Managed Servers continue to write messages to their local server log files. However, by default, when the servers are reconnected, not all the messages written during the disconnected period are forwarded to the domain log file. A Managed Server keeps a specified number of messages in a buffer so they can be forwarded to the Administration Server when the servers are reconnected.

The number of messages kept in the buffer is configured by the `LogMBean` attribute `DomainLogBroadcasterBufferSize`. `DomainLogBroadcasterBufferSize` controls the frequency with which log messages are sent from the Managed Server to the domain server. With the development default of `1`, there is not batching of log messages; only the last logged message is forwarded to the Administration Server domain log. For example, if the Administration Server

is unavailable for two hours and then is restored, the domain log will not contain any messages that were generated during the two hours. See MSI Mode and the Domain Log File in *Administering Server Startup and Shutdown for Oracle WebLogic Server*. In production mode, the default buffer size on the Managed Server is `10`. When the buffer reaches its capacity, the messages in the buffer are flushed by sending them to the domain log on the Administration Server. For performance reasons, it is recommended that you set this value to `10` or higher in production. A higher value will cause the buffer to be broadcast to the domain log less frequently.

If you have configured a value greater than `1`, that number of messages will be forwarded to the domain log when the Managed Server is reconnected to the Administration Server.

> **Note:**
>
> This can result in a domain log file that lists messages with earlier timestamps after messages with later timestamps. When messages from the buffer of a previously disconnected Managed Server are flushed to the Administration Server, those messages are simply appended to the domain log, even though they were generated before the previous messages in the domain log.

Server and Subsystem Logs

Each subsystem within WebLogic Server generates log messages to communicate its status.

For example, when you start a WebLogic Server instance, the Security subsystem writes a message to report its initialization status. To keep a record of the messages that its subsystems generate, WebLogic Server writes the messages to log files.

- Server Log
- Subsystem Logs

Server Log

The server log records information about events such as the startup and shutdown of servers, the deployment of new applications, or the failure of one or more subsystems. The messages include information about the time and date of the event as well as the ID of the user who initiated the event.

You can view and sort these server log messages to detect problems, track down the source of a fault, and track system performance. You can also create client applications that listen for these messages and respond automatically. For example, you can create an application that listens for messages indicating a failed subsystem and sends E-mail to a system administrator.

The server log file is located on the computer that hosts the server instance. Each server instance has its own server log file. By default, the server log file is located in the `logs` directory below the server instance root directory; for example, *`DOMAIN_NAME`*`\servers\`*`SERVER_NAME`*`\logs\`*`SERVER_NAME`*`.log`, where *`DOMAIN_NAME`* is the name of the directory in which you located the domain and *`SERVER_NAME`* is the name of the server.

To view messages in the server log file, you can log on to the WebLogic Server host computer and use a standard text editor, or you can log on to any computer and use the log file viewer in the WebLogic Remote Console. See View Logs in *Oracle WebLogic Remote Console Online Help*.

> **Note:**
>
> Oracle recommends that you do not modify log files by editing them manually. Modifying a file changes the timestamp and can confuse log file rotation. In addition, editing a file might lock it and prevent updates from WebLogic Server, as well as interfere with the Accessor.
>
> For information about the Diagnostic Accessor Service, see Accessing Diagnostic Data With the Data Accessor in *Configuring and Using the Diagnostics Framework for Oracle WebLogic Server*.

In addition to writing messages to a log file, each server instance prints a subset of its messages to standard out. Usually, standard out is the shell (command prompt) in which you are running the server instance. However, some operating systems enable you to redirect standard out to some other location. By default, a server instance prints only messages of a `Notice` severity level or higher to standard out. (A subsequent section, Message Severity describes severity levels.) You can modify the severity threshold so that the server prints more or fewer messages to standard out.

If you use Node Manager to start a Managed Server, the messages that would otherwise be output to `stdout` or `stderr` when starting a Managed Server are instead displayed in the WebLogic Remote Console and written to a single log file for that server instance, *`SERVER_NAME`*`.out`. The server instance's output log is located in the same `logs` directory, below the server instance root directory, along with the WebLogic Server *`SERVER_NAME`*`.log` file; for example, *`DOMAIN_NAME`*`\servers\`*`SERVER_NAME`*`\logs\`*`SERVER_NAME`*`.out`, where *`DOMAIN_NAME`* is the name of the directory in which you located the domain and *`SERVER_NAME`* is the name of the server.

Node Manager writes its own startup and status messages to a single log file, *`NM_HOME`*`/nodemanager.log`, where *`NM_HOME`* designates the Node Manager root directory, by default, *`DOMAIN_HOME`*`/nodemanager`. See Node Manager Configuration and Log Files in *Administering Node Manager for Oracle WebLogic Server*.

Subsystem Logs

The server log messages and log file communicate events and conditions that affect the operation of the server or the application. Some subsystems maintain additional log files to provide an audit of the subsystem's interactions under normal operating conditions. The following list describes each of the additional log files:

- The HTTP subsystem keeps a log of all HTTP transactions in a text file. The default location and rotation policy for HTTP access logs is the same as the server log. You can set the attributes that define the behavior of HTTP access logs for each server or for each virtual host that you define. See Setting Up HTTP Access Logs in *Administering Server Environments for Oracle WebLogic Server*
- Each server has a transaction log which stores information about committed transactions coordinated by the server that may not have been completed. WebLogic Server uses the transaction log when recovering from system crashes or network failures. You cannot directly view the transaction log - the file is in a binary format.

  The Transaction Manager uses the default persistent store to store transaction log files.
- The WebLogic Auditing provider records information from a number of security requests, which are determined internally by the WebLogic Security Framework. The WebLogic

Auditing provider also records the event data associated with these security requests, and the outcome of the requests. Configuring an Auditing provider is optional. The default security realm (myrealm) does not have an Auditing provider configured. See Configuring the WebLogic Auditing Provider in *Administering Security for Oracle WebLogic Server*.

All auditing information recorded by the WebLogic Auditing provider is saved in *WL_HOME**DOMAIN_NAME*\servers*SERVER_NAME*\logs\DefaultAuditRecorder.log. Although an Auditing provider is configured per security realm, each server writes auditing data to its own log file in the server directory.

- The JDBC subsystem records various events related to JDBC connections, including registering JDBC drivers and SQL exceptions. The events related to JDBC are now written to the server log, such as when connections are created or refreshed or when configuration changes are made to JDBC objects. See Monitoring WebLogic JDBC Resources in *Administering JDBC Data Sources for Oracle WebLogic Server*.
- JMS logging is enabled by default when you create a JMS server, however, you must specifically enable it on message destinations in the JMS modules targeted to this JMS server (or on the JMS template used by destinations).

  JMS server log files contain information on basic message life cycle events, such as message production, consumption, and removal. When a JMS destination hosting the subject message is configured with message logging enabled, then each of the basic message life cycle events will generate a message log event in the JMS message log file.

  The message log is located in the logs directory, below the server instance root directory, *DOMAIN_NAME*\servers*SERVER_NAME*\logs\jmsServers*SERVER_NAME*JMSServer\jms.messages.log, where *DOMAIN_NAME* is the name of the directory in which you located the domain and *SERVER_NAME* is the name of the server.

  After you create a JMS server, you can change the default name of its log file, as well as configure criteria for moving (rotating) old log messages to a separate file. See Monitoring JMS Statistics and Managing Messages in *Administering JMS Resources for Oracle WebLogic Server*.

Log Message Format

When a WebLogic Server instance writes a message to the server log file, the first line of each message begins with #### followed by the message attributes. Each attribute is contained between angle brackets.

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

```
####<Sept 22, 2004 10:46:51 AM EST> <Notice> <WebLogicServer> <MyComputer>
<examplesServer><main> <<WLS Kernel>> <> <null> <1080575211904> <BEA-000360> <Server
started in RUNNING mode>
```

In this example, the message attributes are: Locale-formatted Timestamp, Severity, Subsystem, Machine Name, Server Name, Thread ID, User ID, Transaction ID, Diagnostic Context ID, Raw Time Value, Message ID, and Message Text. (A subsequent section, Message Attributes describes each attribute.)

If a message is not logged within the context of a transaction, the angle brackets for Transaction ID are present even though no Transaction ID is present.

If the message includes a stack trace, the stack trace is included in the message text.

WebLogic Server uses the host computer's default character encoding for the messages it writes.

Log File Format Compatibility with Previous WebLogic Server Versions

To configure the logging service to revert to the legacy log format used in earlier versions of WebLogic Server, set the `DomainMBean.LogFormatCompatibilityEnabled` attribute to `true`. In WebLogic Server 12.2.1 and later, the default value of this attribute is `false`.

Format of Output to Standard Out and Standard Error

When a WebLogic Server instance 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.

Here 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.

Message Attributes

The messages for all WebLogic Server instances contain a consistent set of attributes. Table 1-1 lists the server log message attributes. In addition, if your application uses WebLogic logging services to generate messages, its messages also contain these attributes.

Table 1-1 Server Log Message Attributes

Attribute	Description
Locale-formatted Timestamp	Time and date when the message originated, in a format that is specific to the locale. The Java Virtual Machine (JVM) that runs each WebLogic Server instance refers to the host computer operating system for information about the local time zone and format.
Severity	Indicates the degree of impact or seriousness of the event reported by the message. See Message Severity.
Subsystem	Indicates the subsystem of WebLogic Server that was the source of the message; for example, Enterprise Java Bean (EJB) container or Java Messaging Service (JMS).
Machine Name Server Name Thread ID	Identifies the origins of the message: • `Server Name` is the name of the WebLogic Server instance on which the message was generated. • `Machine Name` is the DNS name of the computer that hosts the server instance. • `Thread ID` is the ID that the JVM assigns to the thread in which the message originated. Log messages that are generated within a client JVM do not include these attributes. For example, if your application runs in a client JVM and it uses the WebLogic logging services, the messages that it generates and sends to the WebLogic client log files will not include these attributes.

Table 1-1 (Cont.) Server Log Message Attributes

Attribute	Description
User ID	The user ID under which the associated event was executed. To execute some pieces of internal code, WebLogic Server authenticates the ID of the user who initiates the execution and then runs the code under a special Kernel Identity user ID. Jakartaa EE modules such as EJBs that are deployed onto a server instance report the user ID that the module passes to the server. Log messages that are generated within a client JVM do not include this field.
Transaction ID	Present only for messages logged within the context of a transaction.
Diagnostic Context ID	Context information to correlate messages coming from a specific request or application.
Raw Time Value	The timestamp in milliseconds.
Message ID	A unique six-digit identifier. All message IDs that WebLogic Server system messages generate start with `BEA-` and fall within a numerical range of 0-499999. Your applications can use a Java class called `NonCatalogLogger` to generate log messages instead of using an internationalized message catalog. The message ID for `NonCatalogLogger` messages is always `000000`. See Writing Messages to the WebLogic Server Log in *Adding WebLogic Logging Services to Applications Deployed on Oracle WebLogic Server*.
Message Text	A description of the event or condition.

Message Severity

The severity attribute of a WebLogic Server log message indicates the potential impact of the event or condition that the message reports.Table 1-2 lists the severity levels of log messages from WebLogic Server subsystems, starting from the lowest level of impact to the highest.

Table 1-2 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. See Diagnostic Action Library in *Configuring and Using the Diagnostics Framework for Oracle WebLogic Server*.
`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 1-2 (Cont.) Message Severity

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.

WebLogic Server subsystems generate 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.

If your application uses WebLogic logging services, it can use an additional severity level, `Debug`. See Writing Debug Messages in *Adding WebLogic Logging Services to Applications Deployed on Oracle WebLogic Server*.

Viewing WebLogic Server Logs

The WebLogic Remote Console provides a log viewer for all the log files in a domain.

The log viewer can find and display the messages based on any of the following message attributes: date, subsystem, severity, machine, server, thread, user ID, transaction ID, context ID, timestamp, message ID, or message. It can also display messages as they are logged or search for past log messages.

For information about viewing, configuring, debugging, and filtering message logs, see Log Messages in *Oracle WebLogic Remote Console Online Help*.

For a detailed description of log messages in WebLogic Server message catalogs, see *Error Messages*. This index of messages describes all of the messages emitted by WebLogic subsystems and provides a detailed description of the error, a possible cause, and a recommended action to avoid or fix the error. To view available details, click on the appropriate entry in the Range column (if viewing by range) or the Subsystem column (if viewing by subsystem).

Configuring java.util.logging Logger Levels

WebLogic Server supports configuring `java.util.logging.Logger` levels for named loggers in the JDK `LogManager` from within the WebLogic Server logging configuration.

You can configure `java.util.logging` levels for named loggers using the `PlatformLoggerLevels` attribute in the `LogMBean`. This configuration applies to `java.util.logging.Logger` instances in the JDK's default global `LogManager`.

> **Note:**
>
> This configuration is persisted as part of the WebLogic logging configuration and is not included in the `logging.properties` file.

If your WebLogic domain includes Oracle JRF and is configured to use Oracle Diagnostic Logging (ODL), the `java.util.logging` levels set on the `LogMBean.PlatformLoggerLevels` attribute are ignored. For more information about ODL logging, see Managing Log Files and Diagnostic Data in *Administering Oracle Fusion Middleware*.

To configure WebLogic Server loggers, use the `LoggerSeverities` attribute on the `LogMBean`. See Table 1-2. These loggers are not available in the JDK's default global `LogManager`.

> **Note:**
>
> Log management at application level is supported as of WebLogic Server 14.1.2.0.0 release. You can now declare an application scope in the platform logger, such that the rules apply to a specific application scope only.
> For example, if you set a rule at Platform Logger Levels for `app1:a.b.c` (**Properties Name**) to `FINER` (**Properties Value**); for logger `a.b.c`, a `FINER` rule is applied when logging during `app1`'s execution.

Configuring java.util.logging Logger Levels Using WLST

The following example demonstrates using WLST to configure `java.util.logging` logger levels:

```
wls:/mydomain/serverConfig> edit()
wls:/mydomain/edit> startEdit()
wls:/mydomain/edit !> cd ('/Servers/myserver/Log/myserver')
wls:/mydomain/edit/Servers/myserver/Log/myserver !> props = java.util.Properties()
wls:/mydomain/edit/Servers/myserver/Log/myserver !> props.put("foo.bar","INFO")
wls:/mydomain/edit/Servers/myserver/Log/myserver !>
wls:/mydomain/edit/Servers/myserver/Log/myserver !> cmo.setPlatformLoggerLevels(props)
wls:/mydomain/edit/Servers/myserver/Log/myserver !> save()
Saving all your changes ...
Saved all your changes successfully.
wls:/mydomain/edit/Servers/myserver/Log/myserver !> activate()
Activating all your changes, this may take a while ...
The edit lock associated with this edit session is released once the activation is
completed.
Activation completed
```

2

Configuring WebLogic Logging Services

You can configure the logging output to receive log messages for specific events in Oracle WebLogic Server. Use WebLogic Remote Console, WLST commands or the Java Logging APIs to configure the logging output.
For detailed instructions on filtering and subscribing to messages, see Filtering WebLogic Server Log Messages and Subscribing to Messages.

This chapter describes WebLogic Server logging scenarios and basic configuration tasks:

- Configuration Scenarios
- Overview of Logging Services Configuration
  In the logging process, a logging request is dispatched to subscribed handlers or appenders. Volume control of logging is provided through the `LogMBean` interface.
- Logging Configuration Tasks: Main Steps
  You can configure and filter log messages that the WebLogic Server generates. You can use the WebLogic Remote Console, WebLogic Scripting Tool, or the Java APIs.
- How to Use the Commons API with WebLogic Logging Services
  WebLogic logging services provide the Commons `LogFactory` and `Log` interface implementations that direct requests to the underlying logging implementation being used by WebLogic logging services.
- Rotating Log Files
  The log messages are accumulated in predefined numbered log files. Whenever the file grows in size from the set size, depending on whether it is in development or production mode, the server rotates its server log file.
- Redirecting JVM Output
  The JVM in which a WebLogic Server instance runs sends messages to standard error and standard out. Server as well as application code write directly to these streams instead of using the logging mechanism. However, you can use a configuration option to redirect the JVM output to all registered log destinations, such as the server terminal console and the server log file.
- Redirecting Standard Error and Standard Output
- Preventing Excessive Logging

Configuration Scenarios

WebLogic Server 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.
- Track log information for individual servers in a cluster.

Overview of Logging Services Configuration

In the logging process, a logging request is dispatched to subscribed handlers or appenders. Volume control of logging is provided through the `LogMBean` interface.

WebLogic Server provides handlers for sending log messages to standard out, the server log file, broadcasting messages to the domain log, remote clients, and a memory buffer for tail viewing log events. You can achieve volume control for each type of handler by filtering log messages based on severity level and other criteria. The `LogMBean`, described in *MBean Reference for Oracle WebLogic Server*, defines attributes for setting the severity level and specifying filter criteria for WebLogic Server handlers.

In earlier versions of WebLogic Server, system administrators and developers had only programmatic access to loggers and handlers. In this release of WebLogic Server, you can configure handlers using MBeans, eliminating the need to write code for most basic logging configurations. The WebLogic Server Scripting Tool (WLST) provide an interface for interacting with logging MBeans. Additionally, you can specify `LogMBean` parameters on the command line using `Dweblogic.log.`*`attribute-name=value`*; for example, `Dweblogic.log.StdoutSeverity=Debug`. See Message Output and Logging in *Command Reference for Oracle WebLogic Server*.

For advanced usage scenarios and for configuring loggers, you use the Java Logging APIs.

Setting the severity level on a handler is the simplest type of volume control; for example, any message of a lower severity than the specified threshold severity level, will be rejected. For example, by default, the Stdout Handler has a `Notice` threshold severity level. Therefore, `Info` and `Debug` level messages are not sent to standard out.

Configuring a filter on a handler lets you specify criteria for accepting log messages for publishing; for example, only messages from the HTTP and JDBC subsystems are sent to standard out.

> **Note:**
>
> The `java.util.logging.LoggingPermission` class, described at `http://docs.oracle.com/javase/8/docs/api/java/util/logging/LoggingPermission.html`, is required for a user to change the configuration of a logger or handler. In production environments, we recommend using the Java Security Manager with `java.util.logging.LoggingPermission` enabled for the current user.
>
> See Using the Java Security Manager to Protect WebLogic Resources in *Developing Applications with the WebLogic Security Service*, and the *Java Logging Overview* at `http://docs.oracle.com/javase/8/docs/technotes/guides/logging/overview.html`.

> **Note:**
>
> By default, the query string parameters may show up in the HTTP log files. This allow attackers, access to sensitive data such as passwords, personal information, database details, and so on. To overcome this vulnerability, we have removed the query string as part of the default access log entries.
>
> To restore the query strings to default HTTP access log entires or previous behavior, set this system property `weblogic.servlet.access.log.default.format.with.query` to `TRUE`.

- Using Log Severity Levels
- Using Log Filters

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. WebLogic Server has 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.

For the description of supported severity levels, see `weblogic.logging.Severities` in *Java API Reference for Oracle WebLogic Server*.

You set log levels for handlers and loggers using the WebLogic Remote Console, WLST, or the command line. See Specifying Severity Level for Loggers. Loggers and handlers can also be configured through the API. See Setting the Severity Level for Loggers and Handlers.

Using Log Filters

To provide more control over the messages that a `Logger` object publishes, you can create and set a filter. A filter is a class that uses custom logic to evaluate the log record content which you use to accept or reject a log message; for example, to filter out messages of a certain severity level, from a particular subsystem, or according to specified criteria. The `Logger` object publishes only the log messages that satisfy the filter criteria. You can create separate filters for the messages that each server instance writes to its server log file, standard out, memory buffer, or broadcasts to the domain-wide message log.

You can associate a filter with loggers and handlers. You configure filters for handlers using the WebLogic Remote Console, WLST, or the command line. There are `LogFilterMBean` attributes to define filters for Stdout, Log File, Log Broadcaster, and Memory Handlers, or you can implement custom filtering logic programmatically. The `LogFilterMBean`, described in the *MBean Reference for Oracle WebLogic Server*, defines the filtering criteria based on user ID and subsystem. Filters for loggers are configured only through the API.

See Setting a Filter for Loggers and Handlers.

Logging Configuration Tasks: Main Steps

You can configure and filter log messages that the WebLogic Server generates. You can use the WebLogic Remote Console, WebLogic Scripting Tool, or the Java APIs.

The following steps summarize how you can configure and filter the log messages. Related documentation and later sections in this guide describe these steps in more detail.

1. Use WebLogic Remote Console to configure WebLogic Logging services. See Log Messages in *Oracle WebLogic Remote Console Online Help*.
2. Alternatively, configure log message filtering on the message handler using the WebLogic Scripting Tool. See Configuring Existing Domains in *Understanding the WebLogic Scripting Tool*.
3. Filter log messages published by the logger using the Java APIs. See Filtering Messages by Severity Level or Other Criteria.

How to Use the Commons API with WebLogic Logging Services

WebLogic logging services provide the Commons `LogFactory` and `Log` interface implementations that direct requests to the underlying logging implementation being used by WebLogic logging services.

To use Commons Logging, put the WebLogic-specific Commons classes, *`$WL_HOME`*`/modules/com.bea.core.weblogic.commons.logging_1.3.0.0.jar`, together with the `commons-logging.jar` file in one of the following locations:

- `APP-INF/LIB` or `WEB-INF/LIB` directory
- *`DOMAIN_NAME`*`/LIB` directory
- server `CLASSPATH`

> **Note:**
>
> WebLogic Server does not provide a Commons logging version in its distribution.

Example 2-1 illustrates how to use the Commons interface by setting the appropriate system property.

> **Note:**
>
> When you use the `org.apache.commons.logging.LogFactory` system property to implement the Commons interface as described here, you are implementing it for all application instances running on the server. For information on how to implement Commons logging for specific application instances, without affecting other applications, use the JDK service discovery mechanism or `commons-logging.properties` mechanism to specify the `LogFactory` as described at `http://commons.apache.org/logging/apidocs/org/apache/commons/logging/LogFactory.html#getFactory()`.

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 WebLogic 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://commons.apache.org/logging/apidocs/index.html`.

4. The `weblogic.logging.commons.LogImpl` log methods direct the message to the server log.

Example 2-1 Commons Code Example

```
import org.apache.commons.logging.LogFactory;
import org.apache.commons.logging.Log;

public class MyCommonsTest {
  public void testWLSCommonsLogging() {
    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");
  }
}
```

- Specifying Severity Level for Loggers

Specifying Severity Level for Loggers

WebLogic Server provides a hierarchical Logger tree that lets you specify the Severity level for:

- Generated Message Catalog Logger classes from the XML I18N catalog using `weblogic.i18ngen`.
- Instances of the Commons Logging APIs when the WebLogic Server implementation of the Commons `org.apache.commons.logging.LogFactory` interface is enabled.

All Loggers inherit their Severity level from the nearest parent in the tree. You can, however, explicitly set the Severity level of a Logger, thereby overriding the level that is set for the nearest parent. You can set the Severity level for loggers from the WebLogic Remote Console, WLST, or the command line.

- Specifying Severity Level for WebLogic Server Subsystem Loggers
- Specifying the Severity Level for Commons Logging API Loggers

Specifying Severity Level for WebLogic Server Subsystem Loggers

If you are using the Message Catalog Loggers, the Severity level for messages coming from a specific subsystem are determined by the Severity level of the root Logger. You can override the root Logger Severity level for individual subsystem Loggers such as the DeploymentService Logger, Security Logger, or EJB Logger. For example, suppose the root Logger severity level is `Critical`, and you want to set the Severity Level to `Notice` for the Security subsystem logger and to `Warning` for the EJB subsystem logger. You can do this from the WLST or from the command line:

- Via WLST, use the `set` command to set the value of the `LoggerSeverityProperties` attribute of the `LogMBean`. See Configuring Logging in *Understanding the WebLogic Scripting Tool*.
- From the command line, specify the following parameter in the startup command:

  ```
  -Dweblogic.Log.LoggerSeverityProperties="Security=Notice;EJB=Warning"
  ```

  For a complete index of all subsystem names, see *Error Messages*. The subsystem name is case-sensitive and must be entered exactly as shown in the Subsystem column of the index.

Specifying the Severity Level for Commons Logging API Loggers

If you are using the Commons Logging API, logger names follow the Java package dot notation naming convention. For example, logger names could be `a.b.FooLogger` or `a.b.c.Barlogger`, corresponding to the name of the classes in which they are used. In this case, each dot-separated identifier appears as a node in the Logger tree. For example, there will be a child node named "`a`" under the root Logger, a child node named "`b`" whose parent is "`a`", and so on.

You can configure the Severity for a package or for any Logger at any level in the tree. For example, if you specify the Severity level for package `a.b=Info`, then `Debug` and `Trace` messages coming from all child nodes of package `a.b` will be blocked. You can, however, override the Severity level of a parent node by explicitly setting a value for a child node. For example, if you specify the Severity level for logger `a.b.FooLogger=Debug`, all log messages from `FooLogger` will be allowed, while `Debug` and `Trace` messages will still be filtered for other child nodes under `a.b`.

You can specify the severity level for a package or Logger from the WLST or the command line:

- Via WLST, use the `set` command to set the value of the `LoggerSeverityProperties` attribute of the `LogMBean`. See Configuring Logging in *Understanding the WebLogic Scripting Tool*.
- From the command line, specify the following parameter in the startup command:

  ```
  -Dweblogic.Log.LoggerSeverityProperties="a.b=Info;a.b.FooLogger=Debug"
  ```

Rotating Log Files

The log messages are accumulated in predefined numbered log files. Whenever the file grows in size from the set size, depending on whether it is in development or production mode, the server rotates its server log file.

By default, when you start a WebLogic Server instance in **development mode**, the server automatically renames (rotates) its local server log file as *SERVER_NAME*.log.n. For the remainder of the server session, log messages accumulate in *SERVER_NAME*.log until the file grows to a size of 500 kilobytes.

Each time the server log file reaches this size, the server renames the log file and creates a new *SERVER_NAME*.log to store new messages. By default, the rotated log files are numbered in order of creation *filenamennnnn*, where *filename* is the name configured for the log file. You can configure a server instance to include a time and date stamp in the file name of rotated log files; for example, *server-name-%yyyy%-%mm%-%dd%-%hh%-%mm%*.log.

By default, when you start a server instance in **production mode**, the server rotates its server log file whenever the file grows to 5000 kilobytes in size. It does not rotate the local server log file when you start the server.

You can change these default settings for log file rotation. For example, you can change the file size at which the server rotates the log file or you can configure a server to rotate log files based on a time interval. You can also specify the maximum number of rotated files that can accumulate. After the number of log files reaches this number, subsequent file rotations delete the oldest log file and create a new log file with the latest suffix.

For information about setting up log file rotation, see Rotate Log Files in *Oracle WebLogic Remote Console Online Help*.

To cause the immediate rotation of the server, domain, or HTTP access log file, use the LogRuntime.forceLogRotation() method. See LogRuntimeMBean in *MBean Reference for Oracle WebLogic Server*.

> **Note:**
>
> Though the LogMBean property defines 2 GB as the legal maximum limit for the FileMinsize attribute, WebLogic Server sets a threshold size limit of 500 MB before it forces a hard rotation to prevent excessive log file growth.

The WLST commands in Example 2-2 cause the immediate rotation of the server log file.

Example 2-2 Log Rotation on Demand

```
#invoke WLST
C:\>java weblogic.WLST
#connect WLST to an Administration Server
wls:/offline> connect('username','password')
#navigate to the ServerRuntime MBean hierarchy
wls:/mydomain/serverConfig> serverRuntime()
wls:/mydomain/serverRuntime>ls()
#navigate to the server LogRuntimeMBean
wls:/mydomain/serverRuntime> cd('LogRuntime/myserver')
wls:/mydomain/serverRuntime/LogRuntime/myserver> ls()
-r--   Name                                          myserver
-r--   Type                                          LogRuntime
-r-x   forceLogRotation                              java.lang.Void :
#force the immediate rotation of the server log file
wls:/mydomain/serverRuntime/LogRuntime/myserver> cmo.forceLogRotation()
wls:/mydomain/serverRuntime/LogRuntime/myserver>
```

The server immediately rotates the file and prints the following message:

```
<Mar 2, 2005 3:23:01 PM EST> <Info> <Log Management> <BEA-170017> <The log file
C:\diablodomain\servers\myserver\logs\myserver.log will be rotated. Reopen the
log file if tailing has stopped. This can happen on some platforms like Windows.>
<Mar 2, 2005 3:23:01 PM EST> <Info> <Log Management> <BEA-170018> <The log file
has been rotated to C:\diablodomain\servers\myserver\logs\myserver.log00001. Log
messages will continue to be logged in
C:\diablodomain\servers\myserver\logs\myserver.log.>
```


Specifying the Location of Archived Log Files

By default, the rotated files are stored in the same directory where the log file is stored. You can specify a different directory location for the archived log files by setting the `LogFileRotationDir` property of the `LogFileMBean` from the command line. See `LogFileMBean` in the *MBean Reference for Oracle WebLogic Server*.

The following command specifies the directory location for the archived log files using the `-Dweblogic.log.LogFileRotationDir` Java startup option:

```
java -Dweblogic.log.LogFileRotationDir=c:\foo
-Dweblogic.management.username=installadministrator
-Dweblogic.management.password=installadministrator weblogic.Server
```

Notification of Rotation

When the log file exceeds the rotation threshold that you specify, the server instance prints a log message that states that the log file will be rotated. Then it rotates the log file and prints an additional message that indicates the name of the file that contains the old messages.

For example, if you set up log files to rotate by size and you specify 500K as the minimum rotation size, when the server determines that the file is greater than 500K in size, the server prints the following message:

```
<Sept 20, 2004 1:51:09 PM EST> <Info> <Log Management> <MachineName>
<MedRecServer> <ExecuteThread: '2' for queue: 'weblogic.kernel.System'> <<WLS
Kernel>> <> <> <1095692939895> <BEA-170017> <The log file
C:\Oracle\Middleware\wlserver_12.1\samples\domains\medrec\servers\MedRecServer\logs\medre
c.log will be rotated.
Reopen the log file if tailing has stopped. This can happen on some platforms like
Windows.>
```

The server immediately rotates the file and prints the following message:

```
<Sept 20, 2004 1:51:09 PM EST> <Info> <Log Management> <MachineName>
<MedRecServer> <ExecuteThread: '2' for queue: 'weblogic.kernel.System'>
<<WLS Kernel>> <> <> <1095692939895> <BEA-170018> <The log file has been rotated
to
C:\Oracle\Middleware\wlserver_12.1\samples\domains\medrec\servers\MedRecServer\logs\medre
c.log00001.
Log messages will continue to be logged in
C:\Oracle\Middleware\wlserver_12.1\samples\domains\medrec\servers\MedRecServer\logs\medre
c.log.>
```

Note that the severity level for both messages is `Info`. The message ID for the message before rotation is always `BEA-170017` and the ID for the message after rotation is always `BEA-170018`.

File systems such as the standard Windows file system place a lock on files that are open for reading. On such file systems, if your application is tailing the log file, or if you are using a command such as the DOS `tail -f` command in a command prompt, the tail operation stops after the server has rotated the log file. The `tail -f` command prints messages to standard out as lines are added to a file. For more information, enter `help tail` in a DOS prompt.

To remedy this situation for an application that tails the log file, you can create a JMX listener that notifies your application when the server emits the log rotation message. When your application receives the message, it can restart its tailing operation. To see an example of a JMX listener, see Subscribing to Messages.

Redirecting JVM Output

The JVM in which a WebLogic Server instance runs sends messages to standard error and standard out. Server as well as application code write directly to these streams instead of using the logging mechanism. However, you can use a configuration option to redirect the JVM output to all registered log destinations, such as the server terminal console and the server log file.

When this redirect is enabled, a log entry appears as a message of `Notice` severity. Note that redirecting the JVM output does not capture output from native code; for example, thread dumps from the JVM are not captured.

> **Note:**
>
> Redirecting JVM standard out and standard error messages to the WebLogic logging service by enabling the `LogMBean` attributes, as described in this section, has two key disadvantages you should be aware of:
>
> - JVM messages are redirected asynchronously. In the event of an overload situation, these messages may be dropped.
> - Redirecting JVM messages to the WebLogic logging service in high volume can have a significantly negative impact on system performance and is therefore not recommended.
>
> As a best practice for storing JVM standard out and standard error messages in a log file, Oracle recommends using one of the supported logging APIs instead. Using a logging API ensures that even during times of peak system load, messages are not lost, including the times when those messages are generated in high volume.

- Configuring WebLogic Server to Redirect the JVM Output

Configuring WebLogic Server to Redirect the JVM Output

To configure WebLogic Server to redirect JVM standard out or standard error messages to the WebLogic logging service, you can do one of the following:

- In the `weblogic.Server` command that starts WebLogic Server, include either or both of the following options, as desired:
  - `-Dweblogic.log.RedirectStdoutToServerLogEnabled=true`

    This option redirects JVM *standard out* messages to the WebLogic logging service.
  - `-Dweblogic.log.RedirectStderrToServerLogEnabled=true`

This option redirects JVM *standard error* messages to the WebLogic logging service.

See weblogic.Server Configuration Options in *Command Reference for Oracle WebLogic Server*.

- After the Administration Server has started, you can use the WebLogic Remote Console to redirect the JVM standard out or standard error messages. See Filter Log Messages in *Oracle WebLogic Remote Console Online Help*.
- Use WLST to set either or both of the following attribute values of the `LogMBean` and restart the server:
  - `RedirectStdoutToServerLogEnabled=true`—Redirects the JVM *standard out* messages to the WebLogic logging service.
  - `RedirectStderrToServerLogEnabled=true`—Redirects the JVM *standard error* messages to the WebLogic logging service.

  The WLST commands in the following example redirect the JVM standard out messages in the Administration Server to the server logging destinations.

```
C:\>java weblogic.WLST
wls:/offline> connect('username','password')
wls:/mydomain/serverConfig> edit()
wls:/mydomain/edit> startEdit()
wls:/mydomain/edit !> cd("Servers/myserver/Log/myserver")
wls:/mydomain/edit/Servers/myserver/Log/myserver !>
cmo.setRedirectStdoutToServerLogEnabled(true)
wls:/mydomain/edit/Servers/myserver/Log/myserver !> save()
wls:/mydomain/edit/Servers/myserver/Log/myserver !> activate()
```

  See Navigating MBeans (WLST Online) in*Understanding the WebLogic Scripting Tool*. For more information about the `RedirectStdoutToServerLogEnabled` and `RedirectStderrToServerLogEnabled` attributes, see `LogMBean` in the *MBean Reference for Oracle WebLogic Server*.

Redirecting Standard Error and Standard Output

The `weblogic.RotatingFileRedirector` is a standalone utility tool for redirecting standard error and standard output streams to a rotating log file.Use the following command to run the utility:

```
java weblogic.RotatingFileRedirector [options]
```

The options include:

- `-help`: Prints help about supported options and flags
- `-verbose`: Prints additional output during execution
- `-config Config Properties File`: (Optional) Properties file which specifies the log rotation file parameters as key-value pairs. If not specified, the rotation parameters are defaulted.
- `-configOverride`: Override of a key-value config property pair. This is useful if the same `config.properties` is shared for multiple servers and only the baseLogFileName needs to be different for each server. Multiple overrides can be specified, for example

  ```
  -configOverride baseLogFileName=${SERVER_NAME}.out -configOverride
  rotatedFileCount=10
  ```

The following table lists the properties that can be configured and the default values.

Table 2-1 Properties and Default Values

Property Name	Default Value	Comments

`baseLogFileName`
OR
`baseLogFilePath`
	redirect.log	baseLogFilePath is valid for WebLogic Server versions 12.2.1 and later. Use baseLogFileName for earlier versions.
		Specifies the log file to which stdin will be redirected.

`logFileRotationDir`	null	When not specified rotated log files are created in the same directory as the base log file.
`numberOfFilesLimited`	false	Specifies whether to limit the number of old rotated files on disk.
`bufferSizeKB`	8	Buffer size of the output stream in KB before the contents are flushed to the disk.
`rotateLogOnStartupEnabled`	true	Rotate the log file from previous run if it exists on start up.
`rotatedFileCount`	7	Used in conjunction with `numberOfFilesLimited`. Specifies the number of old rotated logs to keep.
`rotationSize`	500	Size limit when rotation occurs, specified in KB.
`rotationTime`	00:00	Specifies the start time for the rotation when using time based rotation.
`rotationTimeSpan`	24	The interval in hours to rotate the log files. Defaults to 24 hours.
`rotationType`	bySize	Valid values are either bySize or byTime.

Example 2-3 Using the Utility

An example of `config.properties` file contents:

```
rotationSize=100
baseLogFilePath=foo.log
```

The utility is executed as follows:

```
{JAVA_HOME}/bin/java ${JAVA_VM} ${MEM_ARGS} ${JAVA_OPTIONS} -Dweblogic.Name=$
{SERVER_NAME} weblogic.Server 2>$DOMAIN_HOME/logs/mps/${SERVER_NAME}_stderr.log
| ${JAVA_HOME}/bin/java -Xms128m -Xmx256m -cp $WL_HOME/server/lib/weblogic.jar
weblogic.RotatingFileRedirector -configOverride baseLogFilePath=$DOMAIN_HOME/
logs/mps/${SERVER_NAME}_stdout.log -config $DOMAIN_HOME/bt_stdout.prop &
```

Preventing Excessive Logging

Depending on the situation, log messages may become generated at a very high frequency, and often with the same message. This can flood the system with log messages and put excessive load on the system. Excessive logging can occasionally occur due to a number of reasons. For example, a network outage can cause several components to log messages on repeated connection retries, or an incorrect configuration can result in a component emitting log messages repeatedly. Excessive logging can create a number of problems, such as:

- System performance is reduced.
- Log files fill up, and are rotated frequently, increasing the risk of losing useful messages.
- Captured standard out (`stdout`) files grow indefinitely.
- Messages from Managed Servers are broadcast to the domain log, which floods the domain log broadcaster and thereby creating another bottleneck.
- Threads become stuck.

To prevent this problem, the WebLogic logging service provides a feature that monitors the domain for the presence of excessive logging. Log monitoring, which is enabled by default, works by counting the number of messages generated during a specified period of time. If messages are generated at a rate above a set threshold, the logging service inspects individual messages to determine if a specific message is being logged repeatedly. If so, the logging service suppresses, or throttles, that message to reduce the overall rate of logging. Throttling is automatically disabled when the overall message generation volume falls.

A message that is being logged repeatedly is identified by its signature, which consists of the following parameters:

- The logger name that is generating the message
- The message ID
- A portion of the beginning of the message, which is established by the `LogMonitoringThrottleMessageLength` attribute. (The default value is 50, which limits the portion of the message that is evaluated to the first 50 characters.)

To enable log monitoring, configure the following values on the `LogMBean`:

Table 2-2 Attribute

Attribute	Description
`LogMonitoringEnabled={true|false}`	Flag to indicate whether log monitoring is enabled. By default, this value is set to `true`.
`LogMonitoringIntervalSecs=`*`seconds`*	Timer interval, in seconds, during which the number of messages logged is counted. The default is `30`.
`LogMonitoringThrottleThreshold=`*`value`*	Threshold number of messages logged during the specified time interval that either begins or stops message throttling. The default is `1500`.
`LogMonitoringThrottleMessageLength=`*`value`*	Length of the initial portion of the log message that is evaluated during the throttle period. The default is `50`.

Table 2-2 (Cont.) Attribute

Attribute	Description
`LogMonitoringMaxThrottleMessageSignatureCount=`*`value`*	Maximum number of unique message signatures that are monitored during the throttle interval. This value provides a cap on the number of signatures that are stored in an internal cache, which prevents the cache from growing indefinitely and causing an `OutOfMemoryError`.

3

Filtering WebLogic Server Log Messages

Oracle WebLogic Server logging services provide filtering options that give you the flexibility to determine which messages are written to WebLogic Server log files and standard out, and which are written to the log file and standard out that a client JVM maintains. Most of these filtering features are implementations of the Java Logging APIs, which are available in the java.util.logging package.
For related information, see:

- For information about setting up a log filter for a WebLogic Server instance, see Filter Log Messages in *Oracle WebLogic Remote Console Online Help*.
- Subscribing to Messages for information about creating and subscribing a message handler.
- The Role of Logger and Handler Objects
  When WebLogic Server message catalogs and the `NonCatalogLogger` generate messages, they distribute their messages to a `java.util.logging.Logger` object. The `Logger` object publishes the messages to any message handler that has subscribed to the `Logger`.
- Filtering Messages by Severity Level or Other Criteria
- Setting the Severity Level for Loggers and Handlers
  To filter the messages by severity level, you can set the severity level for a `Handler` and `Logger` object using the WLST commands.
- Setting a Filter for Loggers and Handlers
  When you set a filter on the `Logger` object, the filter specifies which messages the object publishes; therefore, the filter affects all handlers that are registered with the `Logger` object as well. When you set a filter on `Handler`, the filter affects only the behavior of the specific handler.

The Role of Logger and Handler Objects

When WebLogic Server message catalogs and the `NonCatalogLogger` generate messages, they distribute their messages to a `java.util.logging.Logger` object. The `Logger` object publishes the messages to any message handler that has subscribed to the `Logger`.

WebLogic Server instantiates `Logger` and `Handler` objects in three distinct contexts. See Figure 3-1 for more details:

- In client JVMs that use WebLogic logging services. This client `Logger` object publishes messages that are sent from client applications running in the client JVM.

  The following handlers subscribe to the `Logger` object in a client JVM:

  – `ConsoleHandler`, which prints messages from the client JVM to the client's standard out.

    If you use the `-Dweblogic.log.StdoutSeverityLevel` Java startup option for the client JVM, WebLogic logging services create a filter for this handler that limits the messages that the handler writes to standard out. See Writing Messages from a Client Application

in *Adding WebLogic Logging Services to Applications Deployed on Oracle WebLogic Server*.

  - `FileStreamHandler`, which writes messages from the client JVM to the client's log file.

- In each instance of WebLogic Server. This server `Logger` object publishes messages that are sent from subsystems and applications that run on a server instance.

  The following handlers subscribe to the server `Logger` object:

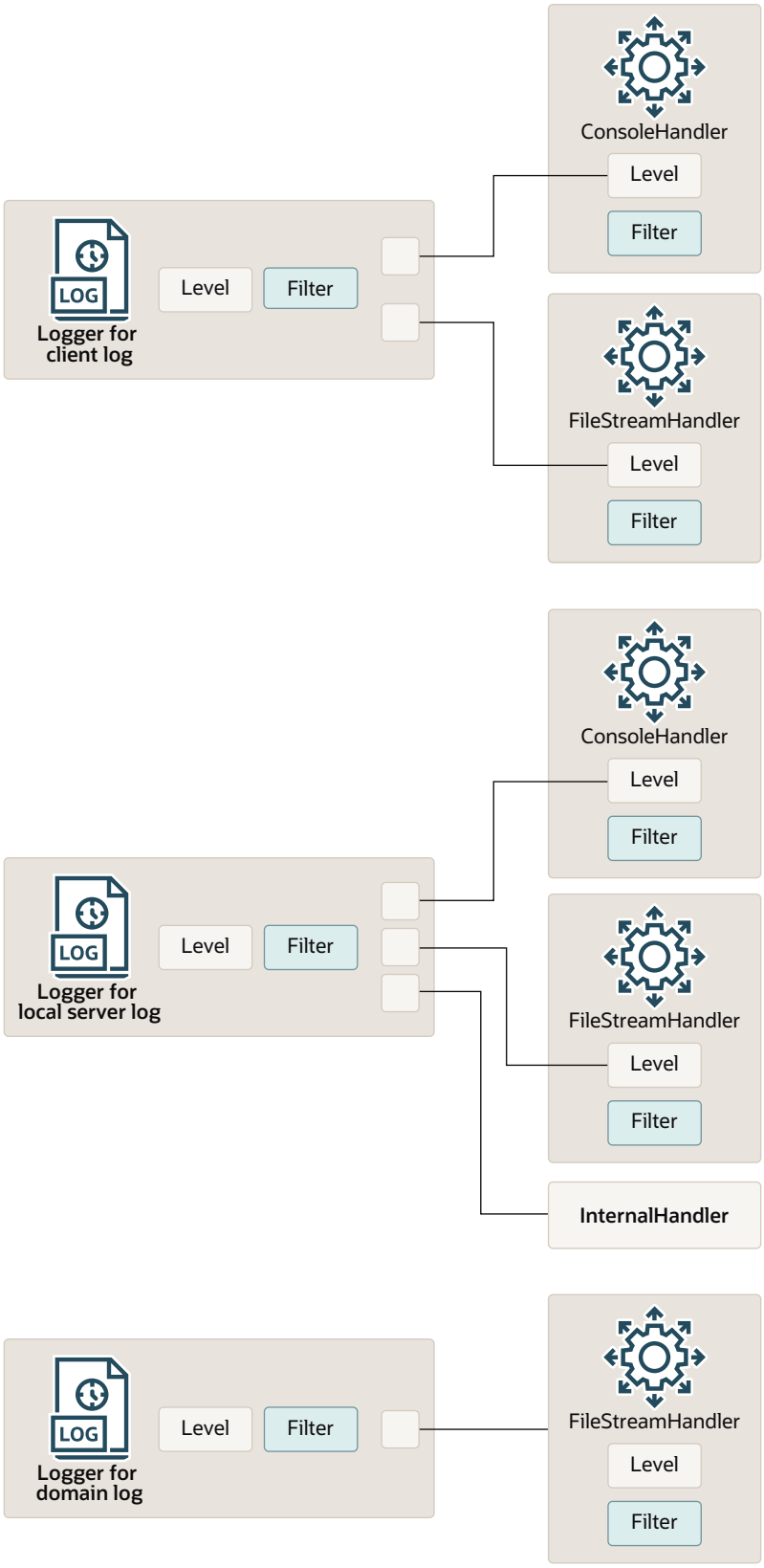
  - `ConsoleHandler`, which makes messages available to the server's standard out.
  - `FileStreamHandler`, which writes messages to the server log file.
  - An internal handler, which broadcasts messages to the domain log and JMX clients, and publishes messages to the Administration Server.

- The Administration Server maintains a domain `Logger` object in addition to a server `Logger` object. The domain `Logger` object receives messages from each Managed Server's `Logger` object.

  The following handler subscribes to the domain `Logger` object:

  - `FileStreamHandler`, which writes messages to the domain log file.

Figure 3-1 WebLogic Logging Services Contexts



Filtering Messages by Severity Level or Other Criteria

When WebLogic Server message catalogs and the `NonCatalogLogger` generate messages, they convert the message severity to a `weblogic.logging.WLLevel` object. A `WLLevel` object can specify any of the following values, from lowest to highest impact:

```
Trace, Debug, Info, Notice, Warning, Error, Critical, Alert, Emergency
```

By default, a `Logger` object publishes messages of all levels. To set the lowest-level message that a `Logger` object publishes, you use a simple `Logger.setLevel` API. When a `Logger` object receives an incoming message, it checks the message level with the level set by the `setLevel` API. If the message level is below the `Logger` level, it returns immediately. If the message level is above the `Logger` level, the `Logger` allocates a `WLLogRecord` object to describe the message.

For example, if you set a `Logger` object level to `Warning`, the `Logger` object publishes only `Warning`, `Error`, `Critical`, `Alert`, or `Emergency` messages.

To provide more control over the messages that a `Logger` object publishes, you can also create and set a filter. A filter is a class that compares data in the `WLLogRecord` object with a set of criteria. The `Logger` object publishes only the `WLLogRecord` objects that satisfy the filter criteria. For example, a filter can configure a `Logger` to publish only messages from the JDBC subsystem. To create a filter, you instantiate a `java.util.logging.Filter` object and use the `Logger.setFilter` API to set it for a `Logger` object.

Instead of (or in addition to) setting the level and a filter for the messages that a `Logger` object publishes, you can set the level and filters on individual message handlers.

For example, you can specify that a `Logger` publishes messages that are of the `Warning` level or higher. Then you can do the following for each handler:

- For the `ConsoleHandler`, set a level and filter that selects only `Alert` messages from the JDBC, JMS, and EJB subsystems. This causes standard out to display only `Alert` messages from the JDBC, JMS, and EJB subsystems.
- For the `FileStreamHandler`, set no additional level or filter criteria. Because the `Logger` object has been configured to publish only messages of the `Warning` level or higher, the log file will contain all messages from all subsystems that are of `Warning` severity level or higher.
- Publish all messages of `Warning` severity level or higher to the domain-wide message log on the Administration Server.

Setting the Severity Level for Loggers and Handlers

To filter the messages by severity level, you can set the severity level for a `Handler` and `Logger` object using the WLST commands.

The WLST commands provide a way to set the severity level for a `Handler` object through standard MBean commands. To set the Severity level for a `Logger` object, you can use the `Logger` API. You can also set the Severity level for a Logger via the WLST or the command line; see Specifying Severity Level for Loggers. To configure Logger and Handler severity level for WLS clients (such as EJB and Web Service clients), you must use the Java Logging API.

- Setting the Level for Loggers
- Setting the Level for Handlers

Setting the Level for Loggers

To set the severity level for a `Logger` object, create a class that does the following:

1. Invokes one of the following `LoggingHelper` methods:
   - `getClientLogger` if the current context is a client JVM.
   - `getServerLogger` if the current context is a server JVM and you want to retrieve the `Logger` object that a server uses to manage its local server log.
   - `getDomainLogger` if the current context is the Administration Server and you want to retrieve the `Logger` object that manages the domain log.

   The `LoggerHelper` method returns a `Logger` object. See the API documentation for the `Logger` class at `http://docs.oracle.com/javase/8/docs/api/java/util/logging/Logger.html`.
2. Invokes the `Logger.setLevel(Level level)` method.

   To set the level of a WebLogic Server `Logger` object, you must pass a value that is defined in the `weblogic.logging.WLLevel` class. WebLogic Server maps the `java.util.logging.Level` to the appropriate `WLLevel`. For a list of valid values, see the description of the `weblogic.logging.WLLevel` class in *Java API Reference for Oracle WebLogic Server*.

   For example:

   ```
   setLevel(WLLevel.Alert)
   ```

Setting the Level for Handlers

To set the severity level for a `Handler` object using the API, create a class that does the following (See Example 3-1):

1. Invokes one of the following `LoggingHelper` methods:
   - `getClientLogger` if the current context is a client JVM.
   - `getServerLogger` if the current context is a server JVM and you want to retrieve the `Logger` object that a server uses to manage its local server log.
   - `getDomainLogger` if the current context is the Administration Server and you want to retrieve the `Logger` object that manages the domain log.

   The `LoggerHelper` method returns a `Logger` object. See the API documentation for the `Logger` class at `http://docs.oracle.com/javase/8/docs/api/java/util/logging/Logger.html`.
2. Invokes the `Logger.getHandlers()` method.

   The method returns an array of all handlers that are registered with the `Logger` object.
3. Iterates through the list of handlers until it finds the `Handler` object for which you want to set a level.

   Use `Handler.getClass().getName()` to determine the type of handler to which the current array index refers.
4. Invokes the `Handler.setLevel(Level level)` method.

To set the level of a WebLogic Server `Handler` object, you must pass a value that is defined in the `weblogic.logging.WLLevel` class. WebLogic Server maps the `java.util.logging.Level` to the appropriate `WLLevel`. For a list of valid values, see the description of the `weblogic.logging.WLLevel` class in *Java API Reference for Oracle WebLogic Server*.

For example:

```
setLevel(WLLevel.Alert)
```

- Example: Setting the Level for Handlers
- Example: Setting the Severity Level for the Stdout Handler

Example: Setting the Level for Handlers

The following example demonstrate how to set level for handlers using API.

Example 3-1 Example: Setting Level for a Handler Object Using the API

```
import java.util.logging.Logger;
import java.util.logging.Handler;
import weblogic.logging.LoggingHelper;
import weblogic.logging.WLLevel;
public class LogLevel {
    public static void main(String[] argv) throws Exception {
        Logger serverlogger = LoggingHelper.getServerLogger();
        Handler[] handlerArray = serverlogger.getHandlers();
        for (int i=0; i < handlerArray.length; i++) {
            Handler h = handlerArray[i];
            if(h.getClass().getName().equals
                      ("weblogic.logging.ConsoleHandler")){
                h.setLevel(WLLevel.Alert);
            }
        }
    }
}
```

Example: Setting the Severity Level for the Stdout Handler

You can configure the severity level for a `Handler` object through the `LogMBean` interface using the command line:

- The WLST commands in Example 3-2 set the severity level for the Stdout Handler to `Info`.

See Using the WebLogic Scripting Tool in *Understanding the WebLogic Scripting Tool*. For more information about `setStdoutSeverity`, see `LogMBean` in *MBean Reference for Oracle WebLogic Server*.

Example 3-2 Setting the Severity Level for the Stdout Handler

```
C:\>java weblogic.WLST
wls:/offline> connect('username','password')
wls:/mydomain/serverConfig> edit()
wls:/mydomain/edit> startEdit()
wls:/mydomain/edit !> cd("Servers/myserver/Log/myserver")
wls:/mydomain/edit/Servers/myserver/Log/myserver !> cmo.setStdoutSeverity("Info")
wls:/mydomain/edit/Servers/myserver/Log/myserver !> save()
wls:/mydomain/edit/Servers/myserver/Log/myserver !> activate()
```

Setting a Filter for Loggers and Handlers

When you set a filter on the `Logger` object, the filter specifies which messages the object publishes; therefore, the filter affects all handlers that are registered with the `Logger` object as well. When you set a filter on `Handler`, the filter affects only the behavior of the specific handler.

The WLST provides a way to set a filter on the `Handler` object through standard MBean commands. To set a filter on the `Logger` object, you must use the `Logger` API. For client-side logging, the only way to set a filter is through using the Java Logging API.

To set a filter:

1. Create a class that implements `java.util.logging.Filter`.

   The class must include the `Filter.isLoggable` method and logic that evaluates incoming messages. If the logic evaluates as true, the `isLoggable` method enables the `Logger` object to publish the message.
2. Place the filter object in the classpath of the JVM on which the `Logger` object is running.
3. To set a filter for a `Logger` object, create a class that does the following:

   Invokes one of the following `LoggingHelper` methods:

   - `getClientLogger` if the current context is a client JVM.
   - `getServerLogger` if the current context is a server JVM and you want to filter the `Logger` object that a server uses to manage its local server log.
   - `getDomainLogger` if the current context is the Administration Server and you want to filter the `Logger` object that manages the domain server log.

   Invokes the `Logger.setFilter(Filter newFilter)` method.
4. To set a filter for a `Handler` object using the API, create a class that does the following:

   Invokes one of the following `LoggingHelper` methods:

   - `getClientLogger` if the current context is a client JVM.
   - `getServerLogger` if the current context is a server JVM and you want to filter the `Logger` object that a server uses to manage its local server log.
   - `getDomainLogger` if the current context is the Administration Server and you want to filter the `Logger` object that manages the domain server log.

   a. Iterates through the list of handlers until it finds the `Handler` object for which you want to set a level.

      Use `Handler.getClass().getName()` to determine the type of handler to which the current array index refers.

   b. Invokes the `Handler.setFilter(Filter newFilter)` method.

The following is an example class that rejects all messages from the Deployer subsystem.

```
import java.util.logging.Logger;
import java.util.logging.Filter;
import java.util.logging.LogRecord;
import weblogic.logging.WLLogRecord;
import weblogic.logging.WLLevel;
public class MyFilter implements Filter {
```

```
    public boolean isLoggable(LogRecord record) {
        if (record instanceof WLLogRecord) {
            WLLogRecord rec = (WLLogRecord)record;
            if (rec.getLoggerName().equals("Deployer")) {
              return false;
            } else {
              return true;
            }
        } else {
          return false;
        }
    }
}
```

You can configure a filter for a `Handler` object through the `LogMBean` interface using the command line:

- The WLST commands in the following example creates and sets a filter on the Domain Log Broadcaster.

```
C:\>java weblogic.WLST
wls:/offline> connect('username','password')
wls:/mydomain/serverConfig> edit()
wls:/mydomain/edit> startEdit()
wls:/mydomain/edit !> cmo.createLogFilter('myFilter')
wls:/mydomain/edit !> cd("Servers/myserver/Log/myserver")
wls:/mydomain/edit/Servers/myserver/Log/myserver !>
cmo.setDomainLogBroadcastFilter(getMBean('/LogFilters/myFilter'))
wls:/mydomain/edit/Servers/myserver/Log/myserver !> save()
wls:/mydomain/edit/Servers/myserver/Log/myserver !> activate()
```

For more information about using WLST, see Using the WebLogic Scripting Tool in *Understanding the WebLogic Scripting tool*. For more information about `setDomainLogBroadcastFilter`, see `LogMBean` in the *MBean Reference for Oracle WebLogic Server*.

- Filtering Domain Log Messages

Filtering Domain Log Messages

To filter the messages that each Managed Server publishes to the domain log, you can create a log filter for the domain log using WLST or WebLogic Remote Console. For information about creating log filters using Remote Console, see Create a Log Filter in *Oracle WebLogic Remote Console Online Help*.

Any Java Logging severity level or filter that you set on the `Logger` object that manages a server instance's log file **supersedes** a domain log filter. For example, if the level of the server `Logger` object is set to `Warning`, a domain log filter will receive only messages of the `Warning` level or higher.

You can define a domain log filter which modifies the set of messages that one or more servers send to the domain log. By default, all messages of severity `Notice` or higher are sent.

> **Note:**
>
> Messages of severity `Debug` are never sent to the domain log, even if you use a filter.

For information about configuring a domain log filter for a WebLogic Server instance using the WebLogic Remote Console, see Filter Log Messages in *Oracle WebLogic Remote Console Online Help*.

4

Subscribing to Messages

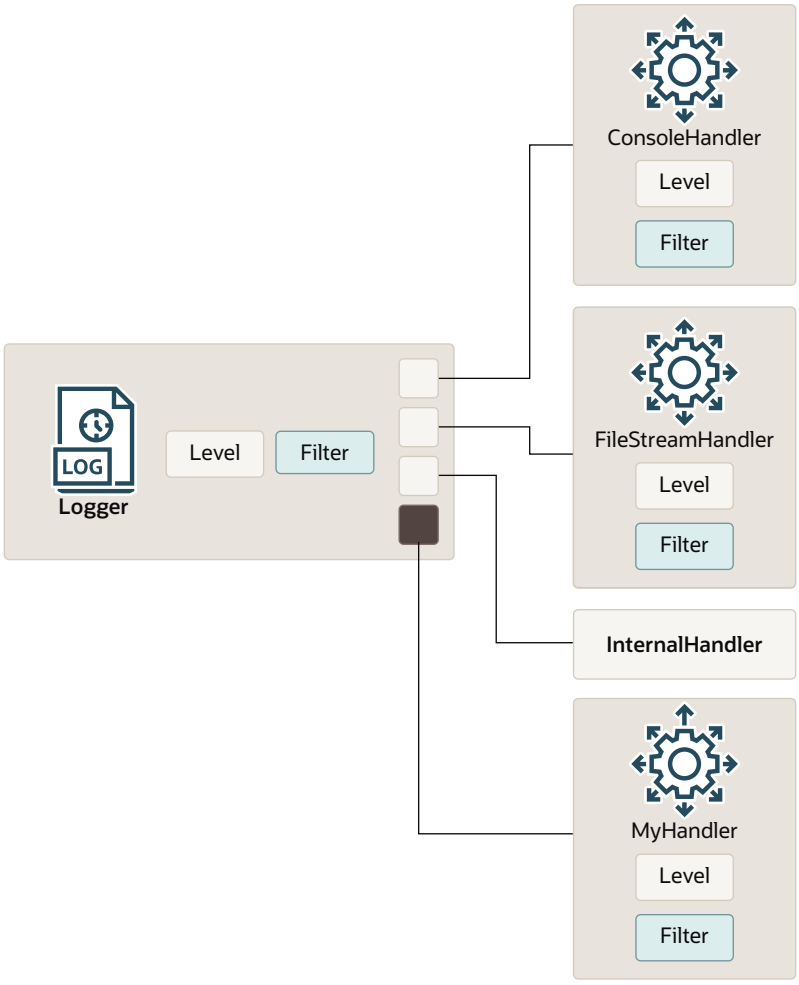
Oracle WebLogic Server logging services provides the ability to create and subscribe a message handler. When WebLogic Server message catalogs and the NonCatalogLogger generate messages, they distribute their messages to a `java.util.logging.Logger` object. The Logger object allocates a WLLogRecord object to describe the message and publishes the WLLogRecord to any message handler that has subscribed to the Logger.

For more information about WebLogic Server loggers and handlers, see The Role of Logger and Handler Objects.

Overview of Message Handlers

WebLogic Server instantiates and subscribes a set of message handlers that receive and print log messages.You can also create your own message handlers and subscribe them to the WebLogic Server `Logger` objects (see Figure 4-1).

Figure 4-1 Subscribing a Handler



For example, if your application runs in a client JVM and you want the application to listen for the messages that your application generates, you can create a handler and subscribe it to the `Logger` object in the client JVM. If your application receives a log message that signals the failure of a specific subsystem, it can perform actions such as:

- E-mail the log message to the WebLogic Server administrator.
- Shut down or restart itself or its subcomponents.

> **Note:**
>
> When creating your own message handlers, be careful to avoid executing custom code which runs in the WebLogic Server process before the server initialization has completed and the server has come to a running state. In some cases, custom code can interfere with server services which are being initialized. For example, custom log handlers that make an outbound RMI call which use the `PortableRemoteObject` before the IIOP server service is initialized, can cause server startup to fail.

Creating and Subscribing a Handler: Main Steps

A handler that you create and subscribe to a `Logger` object receives all messages that satisfy the level and filter criteria of the logger. Your handler can specify additional level and filter criteria so that it responds only to a specific set of messages that the logger publishes.

To create and subscribe a handler:

1. Create a handler class that includes the following minimal set of import statements:

   ```
   import java.util.logging.Handler;
   import java.util.logging.LogRecord;
   import java.util.logging.ErrorManager;
   import weblogic.logging.WLLogRecord;
   import weblogic.logging.WLLevel;
   import weblogic.logging.WLErrorManager;
   import weblogic.logging.LoggingHelper;
   ```

2. In the handler class, extend `java.util.logging.Handler`.
3. In the handler class, implement the `Handler.publish(LogRecord record)` method.

   This method:

   a. Casts the `LogRecord` objects that it receives as `WLLogRecord` objects.

   b. Applies any filters that have been set for the handler.

   c. If the `WLLogRecord` object satisfies the criteria of any filters, the method uses `WLLogRecord` methods to retrieve data from the messages.

   d. Optionally writes the message data to one or more resources.
4. In the handler class, implement the `Handler.flush` and `Handler.close` methods.

   All handlers that work with resources should implement the `flush` method so that it flushes any buffered output and the `close` method so that it closes any open resources.

   When the parent `Logger` object shuts down, it calls the `Handler.close` method on all of its handlers. The close method calls the `flush` method and then executes its own logic.
5. Create a filter class that specifies which types of messages your `Handler` object should receive. See Setting a Filter for Loggers and Handlers.
6. Create a class that invokes one of the following `LoggingHelper` methods:

   - `getClientLogger` if the current context is a client JVM.
   - `getServerLogger` if the current context is a server JVM and you want to attach a handler to the server `Logger` object.
   - `getDomainLogger` if the current context is the Administration Server and you want to attach a handler to the domain `Logger` object.

     `LoggingHelper.getDomainLogger()` retrieves the `Logger` object that manages the domain log. You can subscribe a custom handler to this logger and process log messages from all the servers in a single location.
7. In this class, invoke the `Logger.addHandler(Handler myHandler)` method.
8. Optional. Invoke the `Logger.setFilter(Filter myFilter)` method to set a filter.

Example: Subscribing to Messages in a Server JVM

To subscribe to messages in a server JVM, create a handler that connects to a JDBC data source and issues SQL statements that insert messages into a database table.The example implements the following classes:

- A `Handler` class. See Example: Implementing a Handler Class.
- A `Filter` class. See Setting a Filter for Loggers and Handlers.
- A class that subscribes the handler and filter to a server's `Logger` class. See Example: Subscribing to a Logger Class.

Example: Implementing a Handler Class

The example `Handler` class in Example 4-1 writes messages to a database by doing the following:

1. Extends `java.util.logging.Handler`.
2. Constructs a `javax.naming.InitialContext` object and invokes the `Context.lookup` method to look up a data source named `myPoolDataSource`.
3. Invokes the `javax.sql.DataSource.getConnection` method to establish a connection with the data source.
4. Implements the `setErrorManager` method, which constructs a `java.util.logging.ErrorManager` object for this handler.

   If this handler encounters any error, it invokes the error manager's `error` method. The `error` method in this example:

   a. Prints an error message to standard error.

   b. Disables the handler by invoking `LoggingHelper.getServerLogger().removeHandler(MyJDBCHandler.this)`.

   > **Note:**
   >
   > Instead of defining the `ErrorManager` class in a separate class file, the example includes the `ErrorManager` as an anonymous inner class.

   For more information about error managers, see the API documentation for the `java.util.logging.ErrorManager` class at `http://docs.oracle.com/javase/8/docs/api/java/util/logging/ErrorManager.html`.
5. Implements the `Handler.publish(LogRecord record)` method. The method does the following:

   a. Casts each `LogRecord` object that it receives as a `WLLogRecord` objects.

   b. Calls an `isLoggable` method to apply any filters that are set for the handler. The `isLoggable` method is defined at the end of this handler class.

c. Uses `WLLogRecord` methods to retrieve data from the messages.

   For more information about `WLLogRecord` methods, see the description of the `weblogic.logging.WLLogRecord` class in *Java API Reference for Oracle WebLogic Server*.

d. Formats the message data as a SQL `prepareStatement` and executes the database update.

   The schema for the table used in the example is as follows:

Table 4-1 Schema for Database Table in Handler Example

Name	Null?	Type
MSGID	n/a	CHAR(25)
LOGLEVEL	n/a	CHAR(25)
SUBSYSTEM	n/a	CHAR(50)
MESSAGE	n/a	CHAR(1024)

6. Invokes a `flush` method to flush the connection.
7. Implements the `Handler.close` method to close the connection with the data source.

   When the parent `Logger` object shuts down, it calls the `Handler.close` method, which calls the `Handler.flush` method before executing its own logic.

Example 4-1 illustrates the steps described in this section.

Example 4-1 Implementing a Handler Class

```
import java.util.logging.Handler;
import java.util.logging.LogRecord;
import java.util.logging.Filter;
import java.util.logging.ErrorManager;
import weblogic.logging.WLLogRecord;
import weblogic.logging.WLLevel;
import weblogic.logging.WLErrorManager;
import javax.naming.InitialContext;
import javax.naming.NamingException;
import javax.sql.DataSource;
import java.sql.Connection;
import java.sql.SQLException;
import java.sql.Statement;
import java.sql.PreparedStatement;
import weblogic.logging.LoggingHelper;
public class MyJDBCHandler extends Handler {
   private Connection con = null;
   private PreparedStatement stmt = null;
   public MyJDBCHandler() throws NamingException, SQLException {
      InitialContext ctx = new InitialContext();
      DataSource ds = (DataSource)ctx.lookup("myPoolDataSource");
      con = ds.getConnection();
      PreparedStatement stmt = con.prepareStatement
      setErrorManager(new ErrorManager() {
          public void error(String msg, Exception ex, int code) {
              System.err.println("Error reported by MyJDBCHandler "
                                 + msg + ex.getMessage());
              //Removing any prior istantiation of this handler
              LoggingHelper.getServerLogger().removeHandler(
                                 MyJDBCHandler.this);
```

```
            }
        });
    }
    public void publish(LogRecord record) {
        WLLogRecord rec = (WLLogRecord)record;
        if (!isLoggable(rec)) return;
        try {
            ("INSERT INTO myserverLog VALUES (?, ?, ? ,?)");
            stmt.setEscapeProcessing(true);
            stmt.setString(1, rec.getId());
            stmt.setString(2, rec.getLevel().getLocalizedName());
            stmt.setString(3, rec.getLoggerName());
            stmt.setString(4, rec.getMessage());
            stmt.executeUpdate();
            flush();
        } catch(SQLException sqex) {
            reportError("Error publihsing to SQL", sqex,
                            ErrorManager.WRITE_FAILURE);
        }
    }
    public void flush() {
        try {
            con.commit();
        } catch(SQLException sqex) {
            reportError("Error flushing connection of MyJDBCHandler",
                                sqex, ErrorManager.FLUSH_FAILURE);
        }
    }
     public boolean isLoggable(LogRecord record) {
         Filter filter = getFilter();
         if (filter != null) {
             return filter.isLoggable(record);
         } else {
             return true;
         }
     }
    public void close() {
        try {
            con.close();
        } catch(SQLException sqex) {
             reportError("Error closing connection of MyJDBCHandler",
                                sqex, ErrorManager.CLOSE_FAILURE);
        }
    }
}
```

Example: Subscribing to a Logger Class

The example `Logger` class in Example 4-2 does the following:

1. Invokes the `LoggingHelper.getServerLogger` method to retrieve the `Logger` object.
2. Invokes the `Logger.addHandler(Handler myHandler)` method.
3. Invokes the `Logger.getHandlers` method to retrieve all handlers of the `Logger` object.
4. Iterates through the array until it finds `myHandler`.
5. Invokes the `Handler.setFilter(Filter myFilter)` method.

If you wanted your handler and filter to subscribe to the server's `Logger` object each time the server starts, you could deploy this class as a WebLogic Server startup class.

Example 4-2 Subscribing to a Logger Class

```
import java.util.logging.Logger;
import java.util.logging.Handler;
import java.util.logging.Filter;
import java.util.logging.LogRecord;
import weblogic.logging.LoggingHelper;
import weblogic.logging.FileStreamHandler;
import weblogic.logging.WLLogRecord;
import weblogic.logging.WLLevel;
import java.rmi.RemoteException;
import weblogic.jndi.Environment;
import javax.naming.Context;
public class LogConfigImpl {
    public void configureLogger() throws RemoteException {
        Logger logger = LoggingHelper.getServerLogger();
        try {
            Handler h = null;
            h = new MyJDBCHandler();
            logger.addHandler(h);
            h.setFilter(new MyFilter());
        } catch(Exception nmex) {
            System.err.println("Error adding MyJDBCHandler to logger "
                               + nmex.getMessage());
            logger.removeHandler(h);
        }
    }
    public static void main(String[] argv) throws Exception {
        LogConfigImpl impl = new LogConfigImpl();
        impl.configureLogger();
    }
}
```

Comparison of Java Logging Handlers with JMX Listeners

You can use either Java Logging Handlers or a Java Management Extensions (JMX) listener to receive log messages. You can use both the techniques depending on the requirement. Prior to WebLogic Server 8.1, the only technique for receiving messages from the WebLogic logging services was to create a Java Management Extensions (JMX) listener and register it with a `LogBroadcasterRuntimeMBean`. With the release of WebLogic Server 8.1, you can also use Java Logging handlers to receive (subscribe to) log messages.

While both techniques - Java Logging handlers and JMX listeners - provide similar results, the Java Logging APIs include a `Formatter` class that a `Handler` object can use to format the messages that it receives. JMX does not offer similar APIs for formatting messages. For more information about formatters, see the API documentation for the `Formatter` class at `http://docs.oracle.com/javase/8/docs/api/java/util/logging/Formatter.html`.

In addition, the Java Logging `Handler` APIs are easier to use and require fewer levels of indirection than JMX APIs. For example, the following lines of code retrieve a Java Logging `Logger` object and subscribe a handler to it:

```
Logger logger = LoggingHelper.getServerLogger();
Handler h = null;
h = new MyJDBCHandler();
logger.addHandler(h)
```

To achieve a similar result by registering a JMX listener, you must use lines of code similar to Example 4-3. The code looks up the `MBeanHome` interface, looks up the `RemoteMBeanServer` interface, looks up the `LogBroadcasterRuntimeMBean`, and then registers the listener.

Optimally, you would use Java Logging handlers to subscribe to log messages on your local machine and JMX listeners to receive log messages from a remote machine. If you are already using JMX for monitoring and you simply want to listen for log messages, not to change their formatting or reroute them to some other output, use JMX listeners. Otherwise, use the Java Logging handlers.

Example 4-3 Registering a JMX Listener

```
MBeanHome home = null;
RemoteMBeanServer rmbs = null;
//domain variables
String url = "t3://localhost:7001";
String serverName = "Server1";
String username = "weblogic";
String password = "weblogic";
//Using MBeanHome to get MBeanServer.
try {
    Environment env = new Environment();
    env.setProviderUrl(url);
    env.setSecurityPrincipal(username);
    env.setSecurityCredentials(password);
    Context ctx = env.getInitialContext();
    //Getting the Administration MBeanHome.
    home = (MBeanHome) ctx.lookup(MBeanHome.ADMIN_JNDI_NAME);
    System.out.println("Got the Admin MBeanHome: " + home );
    rmbs = home.getMBeanServer();
} catch (Exception e) {
    System.out.println("Caught exception: " + e);
}
try {
    //Instantiating your listener class.
    MyListener listener = new MyListener();
    MyFilter filter = new MyFilter();
    //Construct the WebLogicObjectName of the server's
    //log broadcaster.
    WebLogicObjectName logBCOname = new
             WebLogicObjectName("TheLogBroadcaster",
           "LogBroadcasterRuntime", domainName, serverName);
    //Passing the name of the MBean and your listener class to the
    //addNotificationListener method of MBeanServer.
    rmbs.addNotificationListener(logBCOname, listener, filter, null);
    } catch(Exception e) {
        System.out.println("Exception: " + e);
    }
}
```