



# BEA WebLogic Server™

## Using WebLogic Logging Services

## Copyright

Copyright © 2002 BEA Systems, Inc. All Rights Reserved.

## Restricted Rights Legend

This software and documentation is subject to and made available only pursuant to the terms of the BEA Systems License Agreement and may be used or copied only in accordance with the terms of that agreement. It is against the law to copy the software except as specifically allowed in the agreement. This document may not, in whole or in part, be copied photocopied, reproduced, translated, or reduced to any electronic medium or machine readable form without prior consent, in writing, from BEA Systems, Inc.

Use, duplication or disclosure by the U.S. Government is subject to restrictions set forth in the BEA Systems License Agreement and in subparagraph (c)(1) of the Commercial Computer Software-Restricted Rights Clause at FAR 52.227-19; subparagraph (c)(1)(ii) of the Rights in Technical Data and Computer Software clause at DFARS 252.227-7013, subparagraph (d) of the Commercial Computer Software--Licensing clause at NASA FAR supplement 16-52.227-86; or their equivalent.

Information in this document is subject to change without notice and does not represent a commitment on the part of BEA Systems. THE SOFTWARE AND DOCUMENTATION ARE PROVIDED "AS IS" WITHOUT WARRANTY OF ANY KIND INCLUDING WITHOUT LIMITATION, ANY WARRANTY OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. FURTHER, BEA Systems DOES NOT WARRANT, GUARANTEE, OR MAKE ANY REPRESENTATIONS REGARDING THE USE, OR THE RESULTS OF THE USE, OF THE SOFTWARE OR WRITTEN MATERIAL IN TERMS OF CORRECTNESS, ACCURACY, RELIABILITY, OR OTHERWISE.

## Trademarks or Service Marks

BEA, Jolt, Tuxedo, and WebLogic are registered trademarks of BEA Systems, Inc. BEA Builder, BEA Campaign Manager for WebLogic, BEA eLink, BEA Manager, BEA WebLogic Commerce Server, BEA WebLogic Enterprise, BEA WebLogic Enterprise Platform, BEA WebLogic Express, BEA WebLogic Integration, BEA WebLogic Personalization Server, BEA WebLogic Platform, BEA WebLogic Portal, BEA WebLogic Server, BEA WebLogic Workshop and How Business Becomes E-Business are trademarks of BEA Systems, Inc.

All other trademarks are the property of their respective companies.

Using WebLogic Logging Services

| <b>Part Number</b> | <b>Document Revised</b> | <b>Software Version</b>            |
|--------------------|-------------------------|------------------------------------|
| N/A                | June 28, 2002           | BEA WebLogic Server<br>Version 7.0 |

---

# Contents

## About This Document

|                                |     |
|--------------------------------|-----|
| Audience.....                  | v   |
| e-docs Web Site.....           | v   |
| How to Print the Document..... | vi  |
| Related Information.....       | vi  |
| Contact Us!.....               | vi  |
| Documentation Conventions..... | vii |

## 1. Overview of WebLogic Logging Services

## 2. Writing Messages to the WebLogic Server Log

|   |      |
|---|------|
| Using the I18N Message Catalog Framework: Main Steps..... | 2-2  |
| Step 1: Create Message Catalogs.....                      | 2-2  |
| Step 2: Compile Message Catalogs.....                     | 2-5  |
| Step 3: Use Messages from Compiled Message Catalogs.....  | 2-6  |
| Using the NonCatalogLogger APIs.....                      | 2-7  |
| Using GenericServlet.....                                 | 2-11 |
| Writing Messages from a Remote Application.....           | 2-11 |
| Writing Messages from a Remote JVM to a File.....         | 2-12 |
| Writing Debug Messages.....                               | 2-12 |

## 3. Viewing the WebLogic Server Logs

## 4. Listening for Messages from the WebLogic Server Log

|  |     |
|--|-----|
| Step 1: Create a Notification Listener.....  | 4-2 |
| Creating a Notification Listener for an Application that Runs Within a<br>WebLogic Server JVM..... | 4-3 |

---

|   |      |
|---|------|
| Creating a Notification Listener for a Remote Application ..... | 4-4  |
| Step 2: Register the Notification Listener .....                | 4-6  |
| Using the addNotificationListener API .....                     | 4-7  |
| Examples for Registering a Notification Listener .....          | 4-8  |
| Step 3: Create and Register a Notification Filter.....          | 4-11 |
| Creating and Registering a Filter.....                          | 4-11 |
| WebLogicLogNotification Objects .....                           | 4-12 |
| Example Notification Filter.....                                | 4-13 |

---

# About This Document

This document describes how your application can write messages to the BEA WebLogic Server™ log files and listen for the log messages that WebLogic Server broadcasts. The document also outlines how you can use the WebLogic Server Administration Console to view log messages.

The document is organized as follows:

- Chapter 1, [Writing Messages to the WebLogic Server Log](#)
- Chapter 2, [Viewing the WebLogic Server Logs](#)
- Chapter 3, [Listening for Messages from the WebLogic Server Log](#)

## Audience

This document is written for application developers who want to build Web applications or other Java 2 Platform, Enterprise Edition (J2EE) components that run on WebLogic Server. It is assumed that readers know Web technologies, object-oriented programming techniques, and the Java programming language.

## e-docs Web Site

BEA product documentation is available on the BEA corporate Web site. From the BEA Home page, click on Product Documentation.

---

# How to Print the Document

You can print a copy of this document from a Web browser, one main topic at a time, by using the File→Print option on your Web browser.

A PDF version of this document is available on the WebLogic Server documentation Home page on the e-docs Web site (and also on the documentation CD). You can open the PDF in Adobe Acrobat Reader and print the entire document (or a portion of it) in book format. To access the PDFs, open the WebLogic Server documentation Home page, click Download Documentation, and select the document you want to print.

Adobe Acrobat Reader is available at no charge from the Adobe Web site at <http://www.adobe.com>.

## Related Information

The BEA corporate Web site provides all documentation for WebLogic Server. Specifically, [Using Log Messages to Manage WebLogic Servers](#) in the *WebLogic Server Administration Guide* describes how to configure log files that a WebLogic Server generates, and the [Internationalization Guide](#) describes how to set up message catalogs that your application can use.

## Contact Us!

Your feedback on BEA documentation is important to us. Send us e-mail at [docsupport@bea.com](mailto:docsupport@bea.com) if you have questions or comments. Your comments will be reviewed directly by the BEA professionals who create and update the documentation.

In your e-mail message, please indicate the software name and version you are using, as well as the title and document date of your documentation. If you have any questions about this version of BEA WebLogic Server, or if you have problems installing and running BEA WebLogic Server, contact BEA Customer Support through BEA

---

WebSupport at <http://www.bea.com>. You can also contact Customer Support by using the contact information provided on the Customer Support Card, which is included in the product package.

When contacting Customer Support, be prepared to provide the following information:

- Your name, e-mail address, phone number, and fax number
- Your company name and company address
- Your machine type and authorization codes
- The name and version of the product you are using
- A description of the problem and the content of pertinent error messages

## Documentation Conventions

The following documentation conventions are used throughout this document.

| <b>Convention</b>    | <b>Item</b>  |
|----------------------|--|
| <b>boldface text</b> | Indicates terms defined in the glossary.                       |
| Ctrl+Tab             | Indicates that you must press two or more keys simultaneously. |
| <i>italics</i>       | Indicates emphasis or book titles.                             |

---

| Convention                             | Item  |
|--|---|
| monospace<br>text                      | <p>Indicates code samples, commands and their options, data structures and their members, data types, directories, and file names and their extensions. Monospace text also indicates text that you must enter from the keyboard.</p> <p><i>Examples:</i></p> <pre>#include &lt;iostream.h&gt; void main ( ) the pointer psz chmod u+w * \tux\data\ap .doc tux.doc BITMAP float</pre> |
| <b>monospace<br/>boldface<br/>text</b> | <p>Identifies significant words in code.</p> <p><i>Example:</i></p> <pre>void <b>commit</b> ( )</pre>   |
| <i>monospace<br/>italic<br/>text</i>   | <p>Identifies variables in code.</p> <p><i>Example:</i></p> <pre>String <i>expr</i></pre>   |
| UPPERCASE<br>TEXT                      | <p>Indicates device names, environment variables, and logical operators.</p> <p><i>Examples:</i></p> <pre>LPT1 SIGNON OR</pre>  |
| { }                                    | <p>Indicates a set of choices in a syntax line. The braces themselves should never be typed.</p>  |
| [ ]                                    | <p>Indicates optional items in a syntax line. The brackets themselves should never be typed.</p> <p><i>Example:</i></p> <pre>buildobjclient [-v] [-o name ] [-f <i>file-list</i>]... [-l <i>file-list</i>]...</pre>   |
|  | <p>Separates mutually exclusive choices in a syntax line. The symbol itself should never be typed.</p>  |

---

---

| Convention | Item   |
|------------|--|
| ...        | <p>Indicates one of the following in a command line:</p> <ul style="list-style-type: none"><li>■ That an argument can be repeated several times in a command line</li><li>■ That the statement omits additional optional arguments</li><li>■ That you can enter additional parameters, values, or other information</li></ul> <p>The ellipsis itself should never be typed.</p> <p><i>Example:</i></p> <pre>buildobjclient [-v] [-o name ] [-f file-list]...<br/>[-l file-list]...</pre> |
| .          | <p>Indicates the omission of items from a code example or from a syntax line.</p> <p>The vertical ellipsis itself should never be typed.</p>   |

---



# 1 Overview of WebLogic Logging Services

The WebLogic Server logging services include facilities for writing, viewing, and listening for log messages. While WebLogic Server subsystems use these services to provide information about events such as the deployment of new applications or the failure of one or more subsystems, your application can also use them to communicate its status and respond to specific events. For example, you can use WebLogic logging services to keep a record of which user invokes specific application components, to report error conditions, or to help debug your application before releasing it to a production environment. In addition, you can configure your application to listen for a log message from a specific subsystem and to respond appropriately.

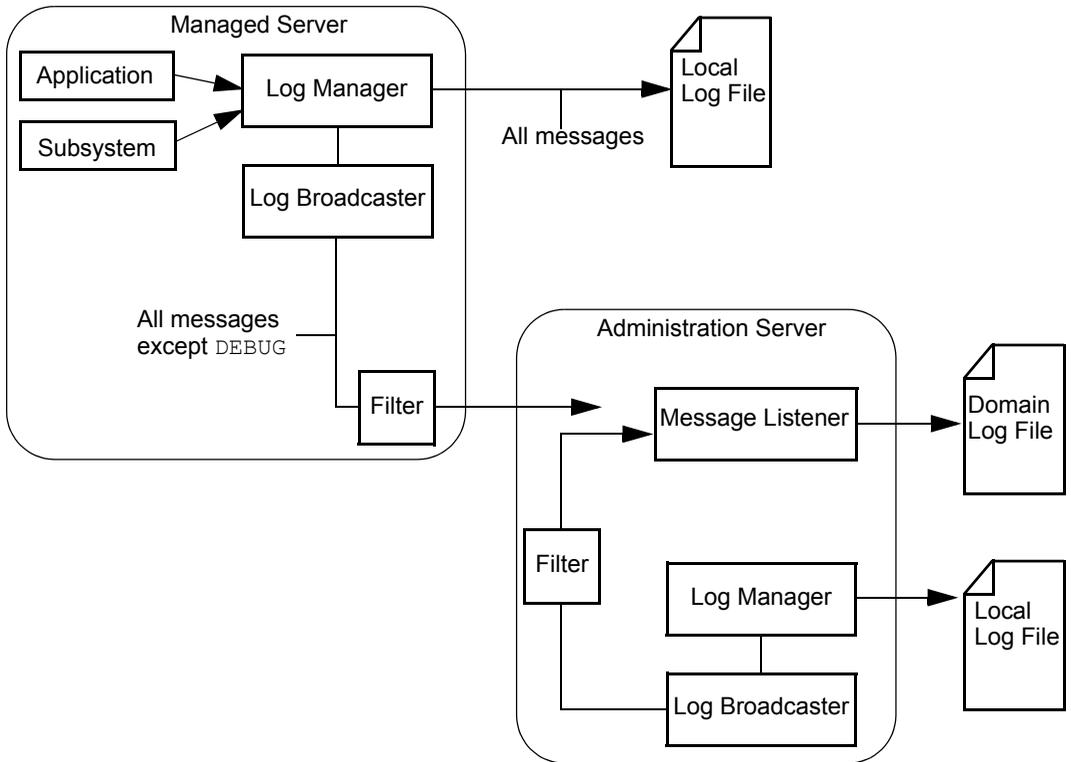
Because each WebLogic Server administration domain can run concurrent, multiple instances of WebLogic Server, the logging services collect messages that are generated on multiple server instances into a single, domain-wide message log. You can use this domain-wide message log to see the overall status of the domain.

To provide this overview of a domain's status, each WebLogic Server instance uses embedded Java Management Extensions (JMX) facilities to broadcast the messages that are in its log files. The broadcast includes all messages that subsystems and applications generate, except for special debug messages that your application can generate. The Administration Server listens for these notifications and writes a subset of them to the domain-wide log file. (See [Figure 1-1](#).)

# 1 Overview of WebLogic Logging Services

---

Figure 1-1 WebLogic Server Logging Services



The remainder of this document describes how your application can write and listen for messages, and how you can view them through the WebLogic Server Administration Console.

# 2 Writing Messages to the WebLogic Server Log

The following sections describe how you can facilitate the management of your application by writing log messages to the WebLogic Server log files:

- [“Using the I18N Message Catalog Framework: Main Steps” on page 2-2](#)
- [“Using the NonCatalogLogger APIs” on page 2-7](#)
- [“Using GenericServlet” on page 2-11](#)

In addition, this section includes the following sections:

- [“Writing Messages from a Remote Application” on page 2-11](#)
- [“Writing Debug Messages” on page 2-12](#)

# Using the I18N Message Catalog Framework: Main Steps

The internationalization (I18N) 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.

To write log messages using the I18N message catalog framework, complete the following tasks:

- [Step 1: Create Message Catalogs](#)
- [Step 2: Compile Message Catalogs](#)
- [Step 3: Use Messages from Compiled Message Catalogs](#)

## Step 1: Create Message Catalogs

A message catalog is an XML file that contains a collection of text messages. Usually, an application uses one message catalog to contain a set of messages in a default language and optional, additional catalogs to contain messages in a local language.

To create and edit a properly formatted message catalog, use the WebLogic Message Editor utility, which is a graphical user interface that is installed with WebLogic Server.

To access the Message Editor, do the following from a WebLogic Server host:

1. Set the classpath by entering `WL_HOME\server\bin\setWLSEnv.cmd` (`setWLSEnv.sh` on UNIX), where `WL_HOME` is the directory in which you installed WebLogic Server.
2. Enter the following command: `java weblogic.MsgEditor`
3. To create a new catalog, choose File→New Catalog.

For more information on using the Message Editor, refer to the following:

- [Using the BEA WebLogic Server Message Editor](#) in the *BEA WebLogic Server Internationalization Guide*.
- [Using Message Catalogs with BEA WebLogic Server](#) in the *BEA WebLogic Server Internationalization Guide*.

When you save your work in the Message Editor, WebLogic Server creates the XML files that comprise your message catalogs. [Listing 2-1](#) provides a sample message catalog that defines three messages. This sample is installed along with WebLogic Server examples in the following location:

```
WL_HOME\samples\server\src\examples\i18n\msgcat\UserServerSCExample.xml.
```

### Listing 2-1 Example Message Catalog

---

```
<?xml version="1.0" encoding="UTF-8"?>

<!DOCTYPE message_catalog PUBLIC "weblogic-message-catalog-dtd"
"http://www.bea.com/servers/wls700/msgcat.dtd">

<message_catalog
  i18n_package="examples.i18n.logging.startup"
  l10n_package="examples.i18n.logging.startup"
  subsystem="UserStartupClass"
  version="1.0"
  baseid="909050"
  endid="909059"
>

<logmessage
  messageid="909050"
  datelastchanged="973906351125"
  datehash="-854388901"
  severity="info"
  method="logInitialMessage(int money 0)"
  stacktrace="false"
>

  <messagebody>
    This message displays a number as currency:
    {0,number,currency}.
  </messagebody>
```

## 2 Writing Messages to the WebLogic Server Log

---

```
<messagedetail>
  Just an example.
</messagedetail>

<cause>
</cause>

<action>
</action>

</logmessage>

<logmessage
  messageid="909051"
  datelastchanged="973906462765"
  datehash="-1800319350"
  severity="warning"
  method="logStringAndPrevCallCount(String str0, int num1)"
  stacktrace="false"
>

  <messagebody>
    This message displays the string "{0}". There
    {1,choice,0#were no previous calls|1#was one previous
    call|2#were {1,number} previous calls} to the logger method
    for this message.
  </messagebody>

  <messagedetail>
</messagedetail>

  <cause>
</cause>

  <action>
</action>

</logmessage>

<logmessage
  messageid="909052"
  datelastchanged="973906532006"
  datehash="-160371672"
  severity="error"
  method="logFinalMessage()"
  stacktrace="false"
>
```

```
<messagebody>
This is not really an error, the example has finished
normally.
</messagebody>

<messagedetail>
</messagedetail>

<cause>
</cause>

<action>
</action>

</logmessage>

</message_catalog>
```

---

## Step 2: Compile Message Catalogs

After you create message catalogs, you use the following utilities to generate Java class files from the XML files:

- The `i18ngen` utility validates message catalogs and generates Java files that you compile and locate in your classpath. Each Java class contains methods that correspond to the messages in the XML file.
- The `l10ngen` utility validates locale-specific catalogs and creates additional properties files for the different locales defined by the catalogs.

To generate and compile the Java class files, do the following:

1. Use `WL_HOME\server\bin\setWLSEnv.cmd` (`setWLSEnv.sh` on UNIX) to set the classpath, where `WL_HOME` is the directory in which you installed WebLogic Server.
2. Enter any of the following commands:
  - `java weblogic.i18ngen [options] files`
  - `java weblogic.i18ngen.tools.l10ngen [options] filelist`

The commands output Java source files. The package name is specified by the `i18n_package` and `l10n_package` attributes in the XML input files.

3. Compile the source files and add them to your classpath.

For complete documentation of the `i18ngen` commands, refer to [Using the BEA WebLogic Server Internationalization Utilities](#) in the *BEA WebLogic Server Internationalization Guide*.

### Step 3: Use Messages from Compiled Message Catalogs

The classes generated by `i18ngen` provide the interface for sending messages to the WebLogic Server log. Within these classes, each log message is represented by a method that your application calls.

For example, for the message catalog named `UserServerSCEExample.xml` (see [Listing 2-1](#)), the `i18ngen` utility generates a class named `examples.i18n.logging.startup.UserServerSCEExampleLogger`. If you want your application to write the `logFinalMessage` message, you call the `UserServerSCEExampleLogger.logFinalMessage()` method. [Listing 2-2](#) illustrates a JSP that calls this method.

#### Listing 2-2 Example JSP That Uses a Message Catalog

---

```
<html>
Order complete. Thanks for your order!

<%@ page
import="examples.i18n.logging.message.UserServerSCEExampleLogger"
%>

<%
    UserServerSCEExampleLogger.logFinalMessage();
%>

</body>
</html>
```

---

---

# Using the NonCatalogLogger APIs

In addition to using the I18N message catalog framework, your application can use the `weblogic.logging.NonCatalogLogger` APIs to send messages to the WebLogic Server log. With `NonCatalogLogger`, instead of calling messages from a catalog, you place the message text directly in your application code. We do not recommend using this facility as the sole means for logging messages if your application needs to be internationalized.

`NonCatalogLogger` is also intended for use by client code that is running in its own JVM (as opposed to running within a WebLogic Server JVM). A subsequent section in this topic, [“Writing Messages from a Remote Application” on page 2-11](#), provides more information.

To use `NonCatalogLogger` in an application that runs within the WebLogic Server JVM, add code to your application that does the following:

1. Imports the `weblogic.logging.NonCatalogLogger` interface.
2. Uses the following constructor to instantiate a `NonCatalogLogger` object:

```
NonCatalogLogger(java.lang.String myApplication)
```

where `myApplication` is a name that you supply to identify messages that your application sends to the WebLogic Server log.

3. Calls any of the `NonCatalogLogger` methods.

Use the following methods to report normal operations:

- `info(java.lang.String msg)`
- `info(java.lang.String msg, java.lang.Throwable t)`

Use the following methods to report a suspicious operation, event, or configuration that does not affect the normal operation of the server/application:

- `warning(java.lang.String msg)`
- `warning(java.lang.String msg, java.lang.Throwable t)`

Use the following methods to report errors that the system/application can handle with no interruption and with limited degradation in service.

- `error(java.lang.String msg)`

## 2 Writing Messages to the WebLogic Server Log

---

- `error(java.lang.String msg, java.lang.Throwable t)`

Use the following methods to provide detailed information about operations or the state of the application. These debug messages are not forwarded to the domain log. If you use this severity level, we recommend that you create a “debug mode” for your application. Then, configure your application to output debug messages only when the application is configured to run in the debug mode. For information about using debug messages, refer to [“Writing Debug Messages” on page 2-12](#).

- `debug(java.lang.String msg)`
- `debug(java.lang.String msg, java.lang.Throwable t)`

All methods that take a `Throwable` argument can print the stack trace in the error log. For information on the `NonCatalogLogger` APIs, refer to the `weblogic.logging.NonCatalogLogger` [Javadoc](#).

[Listing 2-3](#) illustrates a servlet that uses `NonCatalogLogger` APIs to write messages of various severity levels to the WebLogic Server log.

### Listing 2-3 Example NonCatalogLogger Messages

---

```
import java.io.PrintWriter;
import java.io.IOException;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import javax.servlet.ServletException;
import javax.naming.Context;

import weblogic.jndi.Environment;
import weblogic.logging.NonCatalogLogger;

public class MyServlet extends HttpServlet {

    public void service (HttpServletRequest request,
        HttpServletResponse response)
        throws ServletException, IOException {

        PrintWriter out = response.getWriter();
        NonCatalogLogger myLogger = null;

        try {

            out.println("Testing NonCatalogLogger. See WLS Server log for output
                message.");
```

```
// Constructing a NonCatalogLogger instance. All messages from this
// instance will include a <MyApplication> string.
    mylogger = new NonCatalogLogger("MyApplication");

// Outputting an INFO message to indicate that your application has started.
    mylogger.info("Application started.");

// For the sake of providing an example exception message, the next
// lines of code purposefully set an initial context. If you run this
// servlet on a server that uses the default port number (7001), the
// servlet will throw an exception.
    Environment env = new Environment();
    env.setProviderUrl("t3://localhost:8000");

    Context ctx = env.getInitialContext();

}

catch (Exception e){
    out.println("Can't set initial context: " + e.getMessage());
}

// Prints a WARNING message that contains the stack trace.
    mylogger.warning("Can't establish connections. ", e);

}

}

}
```

---

When the servlet illustrated in the previous example runs on a server that specifies a listen port other than 8000, the following messages are printed to the WebLogic Server log file. Note that the message consists of a series of strings, or fields, surrounded by angle brackets (< >).

### Listing 2-4 NonCatalogLogger Output

---

```
####<Jun 26, 2002 12:04:21 PM EDT> <Info> <MyApplication> <peach> <examplesServer>
<ExecuteThread: '10' for queue: 'default'> <kernel identity> <> <000000>
<Application started.>

####<Jun 26, 2002 12:04:23 PM EDT> <Warning> <MyApplication> <peach>
<examplesServer> <ExecuteThread: '10' for queue: 'default'> <kernel identity> <>
<0000000> <Can't establish connections. >
```

## 2 Writing Messages to the WebLogic Server Log

---

```
javax.naming.CommunicationException. Root exception is
java.net.ConnectException: t3://localhost:8000: Destination unreachable; nested
exception is:
```

...

---

[Table 2-1](#) describes all of the fields that `NonCatalogLogger` log messages can contain.

**Table 2-1 NonCatalogLogger Log Message Format**

| Field               | Description  |
|---------------------|--|
| Localized Timestamp | Date and time when message originated, including the year, month, day of month, hours, minutes and seconds. For example, <Jun 26, 2002 12:04:21 PM EDT>.   |
| Severity            | One of the following severity values, which corresponds to the type of method that you used to generate the message:<br>Info, Warning, Error, Debug  |
| Subsystem           | Indicates the source of the message. This is the string that you supply for the <code>NonCatalogLogger</code> constructor.   |
| MachineName         | The name of the computer that hosts the JVM on which your application runs.  |
| ServerName          | The name of the WebLogic Server instance on which your application is running.   |
| ThreadId            | Indicates the execute thread that the current process is using. You can fine-tune an application's access to execute threads (and thereby optimize or throttle its performance) by using multiple execute queues in WebLogic Server. For more information, refer to <a href="#">Using Execute Queues to Control Thread Usage</a> in the <i>BEA WebLogic Server Performance and Tuning</i> guide. |
| User Id             | User on behalf of whom the system was executing when the error was reported.   |
| TransactionId       | Present only for messages logged within the context of a transaction.  |
| Message Id          | A six-digit identifier for the message. The message ID for <code>NonCatalogLogger</code> messages is always 000000.  |
| Message text        | The text that you supply for the <code>NonCatalogLogger</code> method.   |
| ExceptionName       | If the message is logging an Exception, this field contains the name of the Exception.   |

---

# Using GenericServlet

The `javax.servlet.GenericServlet` servlet specification provides the following APIs that your servlets can use to write a simple message to the WebLogic Server log:

- `log(java.lang.String msg)`
- `log(java.lang.String msg, java.lang.Throwable t)`

For more information on using these APIs, refer to the J2EE Javadoc for `javax.servlet.GenericServlet` at <http://java.sun.com/products/servlet/2.3/javadoc/javax/servlet/GenericServlet.html>.

JSPs do not extend from `GenericServlet` and cannot use these APIs. If you want your JSPs to send messages to a log file, consider using the I18N message catalog services or `NonCatalogLogger` APIs.

## Writing Messages from a Remote Application

If your application runs in a JVM that is separate from a WebLogic Server, it can use message catalogs and `NonCatalogLogger`, but the messages are not written to a WebLogic Server log. Instead, the application's messages are written to the remote JVM's standard out.

If you want the WebLogic logging service to send these messages to a log file that the remote JVM maintains, include the following argument in the command that starts the remote JVM:

```
-Dweblogic.log.FileName=logfilename
```

where `logfilename` is the name that you want to use for the remote log file.

If you want a subset of the message catalog and `NonCatalogLogger` messages to standard out as well as the remote JVM log file, include the following additional startup arguments:

```
-Dweblogic.StdoutEnabled=true  
-Dweblogic.StdoutDebugEnabled=boolean  
-Dweblogic.StdoutSeverityLevel = [64 | 32 | 16 | 8 | 4 | 2 | 1 ]
```

where *boolean* is either `true` or `false` and the numeric values for `StdoutSeverityLevel` correspond to the following severity levels:

INFO(64) WARNING(32), ERROR(16), NOTICE(8), CRITICAL(4), ALERT(2) and EMERGENCY(1).

## Writing Messages from a Remote JVM to a File

A remote JVM can generate its own set of messages that communicate information about the state of the JVM itself. For example, you can configure a JVM to generate messages about garbage collection. By default, the JVM sends these messages to standard out. You cannot redirect these messages to the JVM's log file, but you can save them to a separate file. For more information, refer to "[Redirecting System.out and System.err to a File](#)" in the *WebLogic Server Administration Guide*.

## Writing Debug Messages

While your application is under development, you might find it useful to create and use messages that provide verbose descriptions of low-level activity within the application. You can use the `DEBUG` severity level to categorize these low-level messages. All `DEBUG` messages that your application generates are sent to the WebLogic Server log file. (Unlike Log4j, which is a third-party logging service that enables you to dynamically exclude log messages based on level of severity, the WebLogic Server log includes all levels of messages that your application generates.)

You also can configure the WebLogic Server to send `DEBUG` messages to standard out. For more information refer to [Specifying General Log File Settings](#) in the *Administration Console Online Help*.

If you use the `DEBUG` severity level, we recommend that you create a “debug mode” for your application. For example, your application can create an object that contains a boolean value. To enable or disable the debug mode, you toggle the value of the boolean. Then, for each `DEBUG` message, you can create a wrapper that outputs the message only if your application’s debug mode is enabled.

For example:

```
private static boolean debug = Boolean.getBoolean("my.debug.enabled");
if (debug) {
    mylogger.debug("Something debuggy happened");
}
```

You can use this type of wrapper both for messages that use the message catalog framework and that use the `NonCatalogLogger` API.

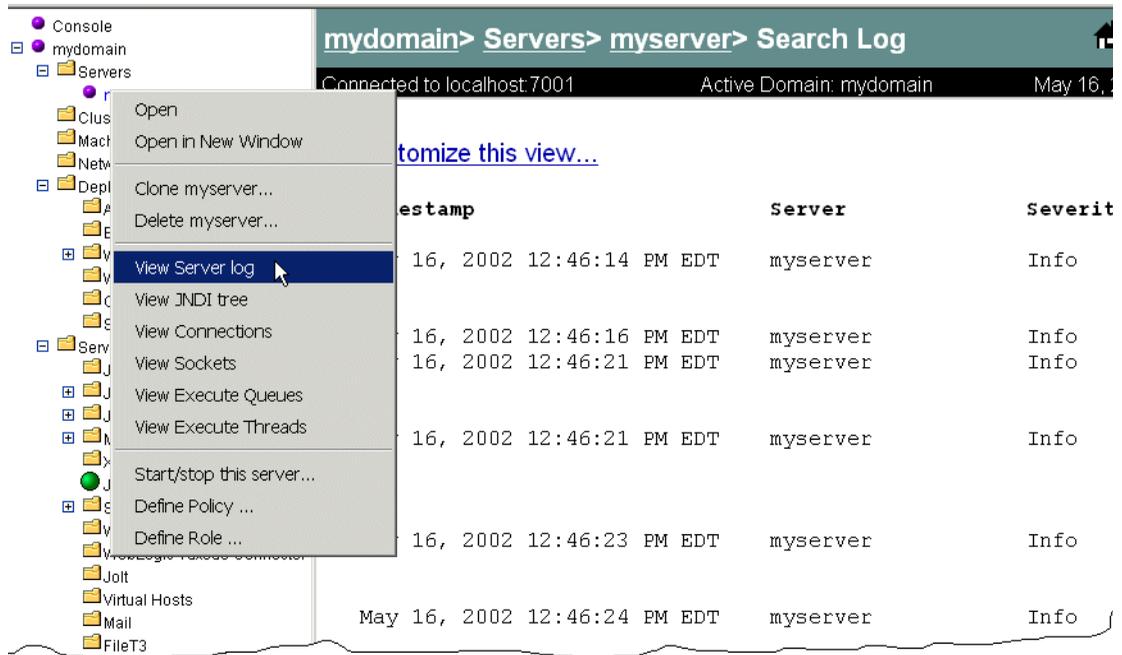


# 3 Viewing the WebLogic Server Logs

The WebLogic Server Administration Console provides separate but similar log viewers for the local server log and the domain-wide message log. The log viewer can search for messages based on fields within the message. For example, it can find and display messages based on the severity, time of occurrence, user ID, subsystem, or the short description. It can also display messages as they are logged, or search for past log messages. (See [Figure 3-1](#).)

### 3 Viewing the WebLogic Server Logs

Figure 3-1 Log Viewer



In addition to viewing messages from the Administration Console, you can specify which messages are sent to standard out. By default, only messages of `ERROR` or higher are sent to standard out.

For information about viewing, configuring, and searching message logs, refer to the following topics:

- [Viewing Server Logs](#) in the Administration Console Online Help
- [Specifying General Log File Settings](#) in the Administration Console Online Help
- [Configuring Debug Information in the Server Log File](#) in the Administration Console Online Help
- [Viewing the Domain Log](#) in the Administration Console Online Help
- [Using Log Messages to Manage WebLogic Servers](#) in the *WebLogic Server Administration Guide*

# 4 Listening for Messages from the WebLogic Server Log

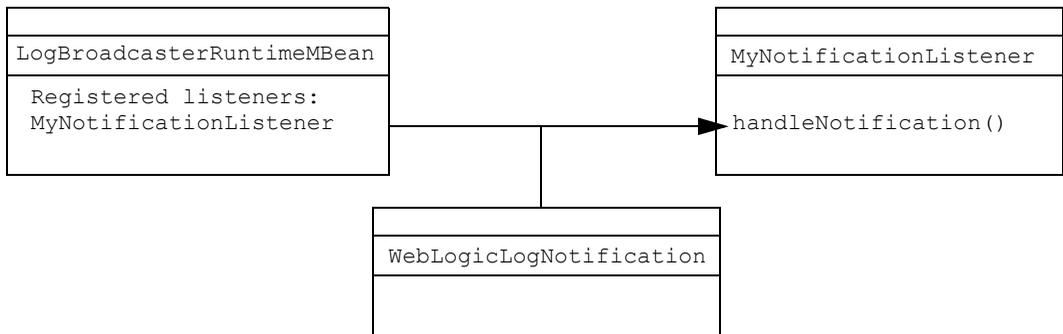
Each WebLogic Server instance broadcasts its log messages in the form of JMX notifications. The broadcast includes all messages (except those of the `DEBUG` severity level) that the WebLogic Server instance, its subsystems, and any applications write to the WebLogic Server log. An Administration Server listens for these notifications and places a subset of them in the domain-wide message log. (See [Figure 1-1](#), “WebLogic Server Logging Services,” on page 1-6.)

Your application also can listen for log messages that are broadcast from a WebLogic Server instance. For example, your application can listen for a log message that signals the failure of a specific subsystem. Then your application can perform actions such as:

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

To listen for these notifications, you create a notification listener and register it with the WebLogic Server broadcast MBean, `LogBroadcasterRuntimeMBean`. A **notification listener** is an implementation of the JMX `NotificationListener` interface. When `LogBroadcasterRuntimeMBean` emits a notification, it uses the registered listener’s `handleNotification` method to pass a `WebLogicLogNotification` object. (See [Figure 4-1](#).)

**Figure 4-1 WebLogic Broadcaster and Your Listener**



A subsequent subsection, “[WebLogicLogNotification Objects](#)” on page 4-12, provides more information about `WebLogicLogNotification` objects.

To enable your application to listen for notifications from a WebLogic Server log, complete the following tasks:

- [Step 1: Create a Notification Listener](#)
- [Step 2: Register the Notification Listener](#)
- [Step 3: Create and Register a Notification Filter](#)

**Note:** If your application runs outside a WebLogic Server JVM, it can listen for WebLogic Server log notifications, but it cannot use WebLogic logging services to broadcast messages.

## Step 1: Create a Notification Listener

The steps that you follow to create a notification listener differ depending on whether your application runs within a WebLogic Server JVM.

This section contains the following subsections:

- [Creating a Notification Listener for an Application that Runs Within a WebLogic Server JVM](#)

- [Creating a Notification Listener for a Remote Application](#)

## Creating a Notification Listener for an Application that Runs Within a WebLogic Server JVM

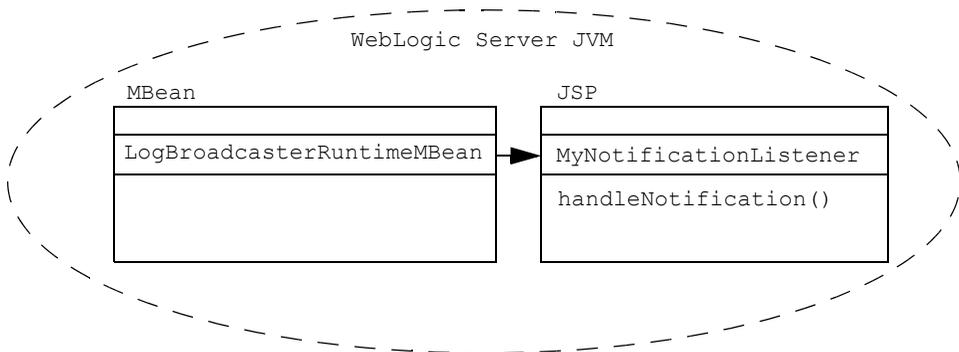
If your application runs within a WebLogic Server JVM, do the following:

1. Import the `javax.management.Notification.*` interfaces. Because WebLogic Server already includes these interfaces and requires them to be on the classpath, you need only to include an import statement in your class.
2. Create a class that implements `NotificationListener`. Your implementation must include the `NotificationListener.handleNotification()` method.

For more information on `NotificationListener`, refer to the `javax.management.Notification` Javadoc, which you can download from <http://jcp.org/aboutJava/communityprocess/final/jsr003/index.html>.

Figure 4-2 shows a system in which a JSP is running within a WebLogic Server JVM. The JSP listens for notifications from `LogBroadcasterRuntimeMBean`.

**Figure 4-2 Listener for a Local JSP**



Listing 4-1 provides an example notification listener for a local client. The listener uses `WebLogicLogNotification` getter methods to print all messages that it receives. For more information, refer to “[WebLogicLogNotification Objects](#)” on page 4-12.

### Listing 4-1 Example Notification Listener for a Local Client

---

```
import javax.management.Notification;
import javax.management.NotificationFilter;
import javax.management.NotificationListener;
import javax.management.Notification.*;

...

public class MyNotificationListener implements
    NotificationListener {

    ...

    public void handleNotification(Notification notification, Object obj) {
        WebLogicLogNotification wln = (WebLogicLogNotification)notification;
        System.out.println("WebLogicLogNotification");
        System.out.println(" type = " + wln.getType());
        System.out.println(" message id = " + wln.getMessageId());
        System.out.println(" server name = " + wln.getServername());
        System.out.println(" timestamp = " + wln.getTimeStamp());
        System.out.println(" message = " + wln.getMessage() + "\n");
    }
}
```

---

## Creating a Notification Listener for a Remote Application

If your application resides outside of the WebLogic Server JVM, do the following:

1. Make sure that `WL_HOME/server/lib/weblogic_sp.jar` and `WL_HOME/server/lib/weblogic.jar` are in the application's classpath.
2. Import the `javax.management.Notification.*` interfaces.
3. Create a class that implements `weblogic.management.RemoteNotificationListener`. `RemoteNotificationListener MBean` makes notifications available to remote applications via RMI by extending `javax.management.NotificationListener` and `java.rmi`.

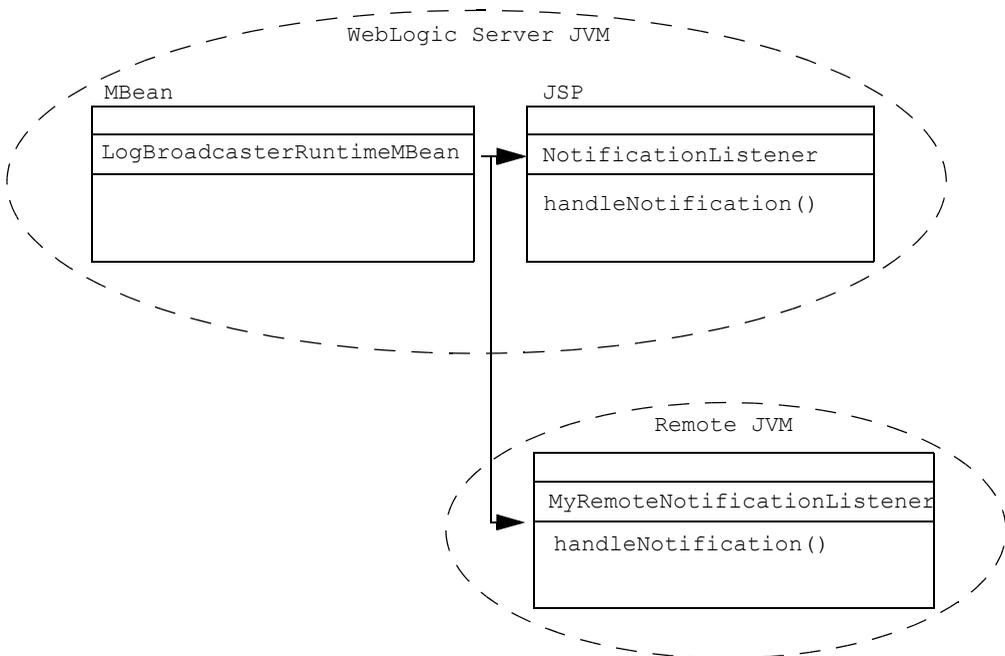
Your implementation must include the

`RemoteNotificationListener.handleNotification()` method. For more information, refer to the

`weblogic.management.RemoteNotificationListener` [Javadoc](#).

Figure 4-3 shows a system in which a JSP runs in the WebLogic Server JVM and an application runs in a remote JVM. To listen for notifications, the JSP implements `NotificationListener` and the remote application implements `RemoteNotificationListener`.

**Figure 4-3 Local JSP and Remote Application**



Listing 4-2 provides an example notification listener for a remote client.

**Listing 4-2 Example Notification Listener for a Remote Client**

```
import javax.management.Notification;
import javax.management.NotificationFilter;
import javax.management.NotificationListener;
import javax.management.Notification.*;

import weblogic.management.RemoteNotificationListener;
import weblogic.management.logging.WebLogicLogNotification;

...
```

## 4 *Listening for Messages from the WebLogic Server Log*

---

```
public class MyRemoteNotificationListener implements
    RemoteNotificationListener {
    ...
    public void handleNotification(Notification notification, Object obj) {
        WebLogicLogNotification wln = (WebLogicLogNotification)notification;
    }
}
```

---

# Step 2: Register the Notification Listener

After you implement your notification listener, you must register it with `LogBroadcasterRuntimeMBean` on a WebLogic Server instance. Because each instance broadcasts its own notifications, you must register your notification listener on each WebLogic Server instance from which you want to receive notifications.

This section describes the code fragment that you use to register a listener. You can add this fragment to a class that runs when your client application starts, when a WebLogic Server instance starts, or whenever you want your application to receive notifications.

To register with the `LogBroadcasterRuntimeMBean` on a WebLogic Server instance, the code must do the following:

1. Import the following interfaces:

```
javax.naming.Context
javax.naming.InitialContext
javax.naming.AuthenticationException
javax.naming.CommunicationException
javax.naming.NamingException
weblogic.jndi.Environment
weblogic.management.MBeanHome
```

2. Obtain the `MBeanServer` from `MBeanHome`. For more information, refer to [Accessing WebLogic Server MBeans](#) in the *Using WebLogic JMX Services Guide*.
3. Use the `addNotificationListener()` method of the `MBeanServer` to register your notification listener with `LogBroadcasterRuntimeMBean`.

## Using the addNotificationListener API

The syntax for the `addNotificationListener` API is as follows:

```
MBeanServer.addNotificationListener(ObjectName name,  
    NotificationListener listener,  
    NotificationFilter filter,  
    java.lang.Object handback)
```

Provide the following values:

- `name` is the object name of the WebLogic Server instance's `LogBroadcasterRuntimeMBean`. You can obtain the object name by doing one of the following:
  - Creating an instance `weblogic.management.WebLogicObjectName`. For more information, refer to the `WebLogicObjectName` [Javadoc](#).
  - Looking up the `weblogic.management.runtime.LogBroadcasterRuntimeMBean` at runtime and calling `.getObjectName()`. For more information, refer to the `LogBroadcasterRuntimeMBean` [Javadoc](#).
  - Using the `weblogic.Admin GET` command. For more information, refer to the [GET command](#) in the *WebLogic Server Administration Guide*.
- `listener` is the instance of the Notification listener you created in “[Step 1: Create a Notification Listener](#)” on page 4-2.
- `filter` is a filter object. If `filter` is `null`, no filtering will be performed before handling notifications. The next section, “[Step 3: Create and Register a Notification Filter](#)” on page 4-11, describes creating and registering a filter object.
- `handback` is the context to be sent to the listener when a notification is broadcast.

Complete documentation for the `addNotificationListener` API is available in the Javadoc for `javax.management.MBeanServer`, which you can download from <http://jcp.org/aboutJava/communityprocess/final/jsr003/index.html>.

# Examples for Registering a Notification Listener

The following examples register the listener defined in [Step 1: Create a Notification Listener](#). The examples in [Listing 4-3](#) and [Listing 4-4](#) do the following:

1. Use the `weblogic.management.Helper` API to obtain the server-specific `MBeanHome` interface for a server named `peach`. For more information about obtaining the `MBeanHome` interface, refer to [Accessing WebLogic Server MBeans](#) in the *Programming WebLogic Management Services with JMX* guide.
2. Use the `MBeanHome` interface to retrieve the corresponding `MBeanServer` interface.
3. Use a different method for retrieving the `LogBroadcasterRuntimeMBean` object name.
4. Instantiate the listener object defined in [Step 1: Create a Notification Listener](#).
5. Register the listener object with the `LogBroadcasterRuntimeMBean`.

[Listing 4-3](#) uses `WebLogicObjectName` to construct the `LogBroadcasterRuntimeMBean` object name.

### Listing 4-3 Using WebLogicObjectName

---

```
public void find(String host,
                int port,
                String username
                String password){

    String url = "t3://" + host +
                ":" + port;

    //Get the server's MBeanHome interface.
    try {
        serverSpecificHome = (MBeanHome)Helper.getMBeanHome(username,
                                                            password,
                                                            url,
                                                            peach);
    } catch (IllegalArgumentException iae) {
        System.out.println("Illegal Argument Exception: " + iae);
    }
}
```

## Step 2: Register the Notification Listener

---

```
//Use MBeanHome to get the server's MBeanServer interface.
MBeanServer mServer = serverSpecificHome.getMBeanServer();

//Construct the WebLogicObjectName of the server's LogBroadcasterRuntimeMBean.
WebLogicObjectName logBCOname = new WebLogicObjectName("WebLogicLogBroadcaster",
    "LogBroadcasterRuntime",
    myDomain,
    myServer);

//Instantiate a listener object.
MyRemoteNotificationListener myListener = new MyRemoteNotificationListener();

//Register the listener.
mServer.addNotificationListener( logBCOname,
    myListener,
    null,
    null);
}
```

---

**Listing 4-4** uses `MBeanHome.getMBeanByClass` to retrieve the `LogBroadcasterRuntimeMBean` object name.

### **Listing 4-4 Using getObjectNames()**

---

```
public void find(String host,
    int port,
    String username
    String password){

    String url = "t3://" + host +
        ":" + port;

//Get the server's MBeanHome interface.
    try {
        serverSpecificHome = (MBeanHome)Helper.getMBeanHome(username,
            password,
            url,
            peach);
    } catch (IllegalArgumentException iae) {
        System.out.println("Illegal Argument Exception: " + iae);
    }

//Use MBeanHome to get the server's MBeanServer interface.
    MBeanServer mServer = serverSpecificHome.getMBeanServer();
```

## 4 *Listening for Messages from the WebLogic Server Log*

---

```
//Use getMBeanByClass to retrieve the object name.
LogBroadcasterRuntimeMBean logBCOname = (LogBroadcasterRuntimeMBean)
    home.getMBeanByClass(Class.forName
        ("weblogic.management.runtime.LogBroadcasterRuntimeMBean")
    );

//Instantiate a listener object.
MyRemoteNotificationListener myListener = new MyRemoteNotificationListener();

//Register the listener.
mServer.addNotificationListener( logBCOname,
    myListener,
    null,
    null);
}
```

---

**Listing 4-5** assumes that you used `weblogic.Admin GET` to retrieve the `LogBroadcasterRuntimeMBean` object name. It also illustrates the format of object names that `weblogic.Admin GET` returns.

### **Listing 4-5 Using weblogic.Admin GET**

---

```
MyRemoteNotificationListener myListener = new MyRemoteNotificationListener();
MBeanServer mServer = home.getMBeanServer();

ObjectName logBCOname = new
ObjectName("mydomain:Location=myserver,Name=TheLogBroadcaster,Type=LogBroadcast
erRuntime");

mServer.addNotificationListener( logBCOname,
    myListener,
    null,
    null);
```

---

## Step 3: Create and Register a Notification Filter

By default, the notification listener that you registered in the previous section listens for all notifications from `LogBroadcasterRuntimeMBean` and sends them to your application. You can configure the `LogBroadcasterRuntimeMBean` to send only the notifications that are pertinent to your application by creating and registering a filter. The filter determines whether a notification matches a set of criteria that you create, and the `LogBroadcasterRuntimeMBean` sends the notification only if the filter evaluates as true.

This section contains the following subsections:

- [Creating and Registering a Filter](#)
- [WebLogicLogNotification Objects](#)
- [Example Notification Filter](#)

### Creating and Registering a Filter

To create and register a filter, do the following:

1. Import the following interfaces:

```
import javax.management.Notification
import javax.management.NotificationFilter
import javax.management.Notification.*
```

2. Create a serializable object that does the following:
  - a. Implements `javax.management.NotificationFilter`.
  - b. Searches a notification for a string.

To search a notification that has been cast as a `WebLogicLogNotification` object, you can use `WebLogicLogNotification` getter methods. For example, you can use the getter methods to get the message timestamp, severity, user ID, the name of the subsystem that generated the message, the

message text, and other data. For more information, refer to [WebLogicLogNotification Objects](#).

- c. Uses a boolean to indicate whether the serializable object returns a true value.
  - d. (Optional) Includes code that carries out an action depending on the value of the boolean. For example, your filter can use the JavaMail API to send e-mail to an administrator if a message is of severity `WARNING` or higher.
3. Use the `addNotificationListener` API to register the filter. For more information, refer to [“Using the addNotificationListener API” on page 4-7](#).

## WebLogicLogNotification Objects

All messages that a WebLogic Server generates are cast as `weblogic.management.logging.WebLogicLogNotification` objects. `WebLogicLogNotification` objects contain the following fields:

- **Type**—identifies the notification as required by the JMX specification. This field has the format:

```
weblogic.logMessage.subSystem.messageID
```

where `subSystem` indicates the subsystem or application that issued the log message this notification contains, and `messageID` indicates the internal WebLogic Server message ID.

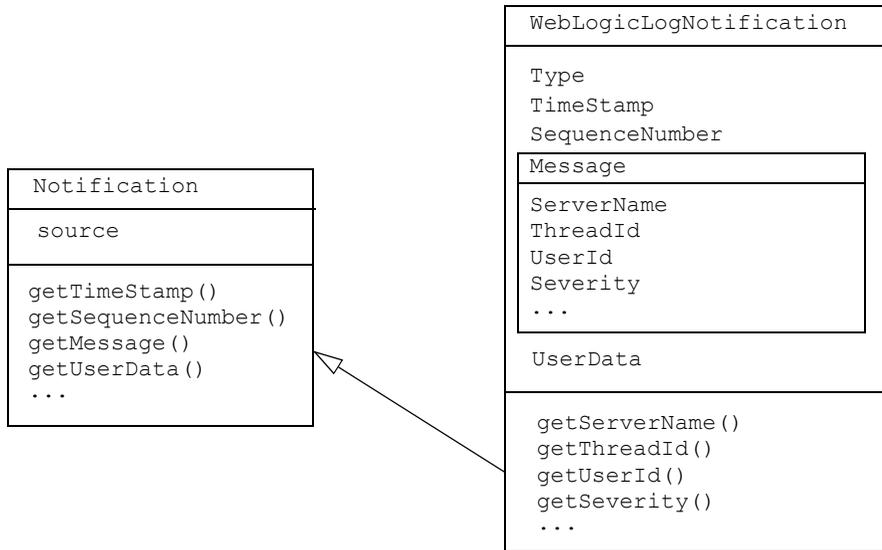
**Note:** For `NonCatalogLogger` messages, the message ID is always 000000.

- **Time stamp**—indicates the time at which the log message causing this notification was generated by the server.
- **Sequence number**.
- **Message**—contains the log message.
- **User data**—the user data field is not currently used.

A `WebLogicLogNotification` inherits getter methods from `javax.management.Notification` and it provides one getter method for each field within the log message. (See [Figure 4-4](#).)

You can use these getter methods to search or print the information within the `WebLogicLogNotification`. For more information, refer to the `weblogic.management.logging.WebLogicLogNotification` [Javadoc](#).

**Figure 4-4 WebLogicLogNotification Getter Methods**



## Example Notification Filter

[Listing 4-6](#) provides an example `NotificationFilter` that uses the `WebLogicLogNotification.getType` method.

**Listing 4-6 Example Notification Filter**

---

```
import javax.management.Notification;
import javax.management.NotificationFilter;
import javax.management.Notification.*;
import weblogic.management.logging.WebLogicLogNotification;
```

## 4 *Listening for Messages from the WebLogic Server Log*

---

```
.....

public class MyLogNotificationFilter implements NotificationFilter,
    java.io.Serializable {

public MyLogNotificationFilter() {
    subsystem = "";
}

public boolean isNotificationEnabled(Notification notification) {
    if (!(notification instanceof WebLogicLogNotification)) {
        return false;
    }

    WebLogicLogNotification wln = (WebLogicLogNotification)notification;

    if (subsystem == null ||
        subsystem.equals("")) {
        return true;
    }

    StringTokenizer tokens = new StringTokenizer(wln.getType(), ".");
    tokens.nextToken();
    tokens.nextToken();
    return (tokens.nextToken().equals(subsystem));
}

public void setSubsystemFilter(String newSubsystem) {
    subsystem = newSubsystem;
}

}
```

---

---

# Index

## A

- `addNotificationListener` method 4-7
- Administration Console 3-1
- Administration Server 1-5, 4-1
- ALERT severity level 2-12
- arguments for starting a remote JVM 2-11

## B

- broadcasts
  - from remote applications 4-2
  - from WebLogic Servers 4-1, 4-6

## C

- catalogs, message 2-2–2-6
- class files 2-5
- classpath 2-2, 4-4
- client applications. *See* remote applications
- client JVMs. *See* remote JVMs
- compiling Java class files 2-5
- Console, Administration 3-1
- CRITICAL severity level 2-12
- customer support contact information vi

## D

- debug messages 1-5, 2-12, 2-13, 4-1
- documentation, where to find it v

## E

- e-mail 4-1, 4-12
- EMERGENCY severity level 2-12
- environment, setting 2-2
- ERROR severity level 2-12
- examples
  - message catalog 2-3
  - `NonCatalogLogger` message 2-8
  - notification filter 4-13
  - notification listener for a local application 4-4
  - notification listener for a remote application 4-5
  - registering notification listeners 4-8
  - using a message catalog 2-6
- `ExceptionName` message field 2-10
- excluding log messages 2-12

## F

- filters. *See* notification filters
- format of messages. *See* message format

## G

- garbage collection 2-12
- `GenericServlet` 2-11
- getter methods for
  - `WebLogicLogNotification` object 4-13

---

## H

`handleNotification` method  
    defined 4-1  
    for local applications 4-3  
    for remote applications 4-4

## I

INFO severity level 2-12  
interfaces, importing  
    for `NonCatalogLogger` APIs 2-7  
    for notification filters 4-11  
    for notification listeners 4-3–4-6  
internationalization, recommendations 2-2,  
    2-7

## J

Java class files 2-5  
Java package names 2-5  
Java Virtual Machines. *See* remote JVMs  
JMX 1-5  
JMX notifications. *See* notifications  
JSPs  
    `GenericServlet` 2-11  
    message catalogs 2-6  
    `NotificationListeners` 4-3, 4-5

## L

listeners. *See* notification listeners  
localization  
    properties files 2-5  
    recommendations 2-2  
`LocalizedTimestamp` message field 2-10  
location of messages for remote applications  
    2-11  
log files  
    for domains 4-1  
    for remote applications 2-11  
    for remote JVMs 2-11

log message format  
    message catalog 2-2  
    `NonCatalogLogger` 2-10  
log message ID  
    example 2-4  
    for `NonCatalogLogger` messages 4-12  
log message text  
    example 2-4  
    filtering 4-11  
    message catalog 2-2  
    `NonCatalogLogger` 2-7, 2-10  
    searching 4-12  
log messages  
    broadcasting. *See* broadcasts  
    excluding debug 2-12  
    from servlets 2-11  
    printing 4-3  
    searching in the log viewer 3-1  
    *See also* notifications  
    viewing 3-1  
log messages, broadcasting. *See* broadcasts  
log viewers 3-1  
Log4j 2-12  
`LogBroadcasterRuntimeMBean` object  
    defined 4-1  
    obtaining object name 4-7

## M

`MachineName` message field  
    defined 2-10  
    getting 4-11  
`MBeanHome` interface 4-6  
`MBeanServer` interface 4-6  
message catalogs 2-2–2-6  
Message Editor GUI 2-2  
`Message` message field 2-10, 4-12  
`MessageId` message field  
    defined 2-10  
    getting 4-11  
messages. *See* log messages

---

millisecondsFromEpoch message field  
2-10

## N

NonCatalogLogger object

APIs 2-7

example 2-8

message format 2-10

message IDs 4-12

recommendations 2-7

NOTICE severity level 2-12

notification filters

creating and registering 4-11

defined 4-11

example 4-13

specifying in the

addNotificationListener

method 4-7

notification listeners

default behavior 4-11

defined 4-1

example of registering 4-8

for a local application, example 4-4

for a remote application, example 4-5

registering 4-6

starting 4-6

notifications

broadcasting defaults 4-11

defined 4-1

*See also* messages

## O

object name for

LogBroadCasterRuntimeMBean

objects 4-7–4-10

options for starting a remote JVM 2-11

## P

package names 2-5

parameters for starting a remote JVM 2-11

path, setting. *See also* classpath 2-2

printing messages 4-3

printing product documentation vi

properties files for localization 2-5

## R

remote applications 2-11

remote JVMs

log files 2-11

NonCatalogLogger messages 2-7

notification listeners 4-4

startup arguments 2-11

writing to standard out 2-11

RemoteNotificationListener object 4-4

RMI 4-4

## S

SequenceNumber message field

defined 4-12

getting 4-11

ServerName message field

defined 2-10

getting 4-11

servlets 2-11

severity levels

defined 2-12

numerical values 2-12

using to exclude messages 2-12

Severity message field

defined 2-10

getting 4-11

standard out 2-11, 3-2

starting notification listeners 4-6

startup arguments 2-11

Subsystem message field 2-10

support

---

technical vi

## T

ThreadId message field

defined 2-10

getting 4-11

Throwable message field 4-11

TimeStamp message field

defined 4-12

getting 4-11

TransactionId message field

defined 2-10

getting 4-11

transmitting messages. *See* broadcasts

Type message field

defined 4-12

getting 4-11

## U

UserData message field 4-12

UserId message field 2-10

## V

viewing message logs 3-1

## W

WARNING severity level 2-12

weblogic.Admin GET command 4-7

weblogic.MsgEditor command 2-2

WebLogicLogNotification object

example of using 4-3

getter methods 4-13

searching 4-12

## X

XML 2-2, 2-5