



BEA WebLogic Server™

Internationalization Guide

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.

Internationalization Guide

Part Number	Document Revised	Software Version
N/A	August 20, 2002	BEA WebLogic Server Version 7.0

Contents

About This Document

Audience.....	viii
e-docs Web Site.....	viii
How to Print the Document.....	viii
Related Information.....	ix
Contact Us!.....	ix
Documentation Conventions.....	x

1. Overview of Internationalization for WebLogic Server

About Internationalization and Localization Standards.....	1-1
Understanding Localization for WebLogic Server.....	1-2
Understanding Message Catalogs.....	1-3
Understanding Java Interfaces for Internationalization.....	1-4
Main Steps for Creating an Internationalized Message.....	1-4

2. Using Message Catalogs with BEA WebLogic Server

Overview of Message Catalogs.....	2-1
Message Catalog Hierarchy.....	2-2
Choosing Names for Message Catalogs.....	2-3
Using Message Arguments.....	2-4
Message Catalog Formats.....	2-5
Elements of a Log Message Catalog.....	2-5
message_catalog.....	2-6
log_message.....	2-7
Other log_message Catalog Elements.....	2-8
Log Message Catalog Example.....	2-9
Elements of a Simple Text Message Catalog.....	2-10

message_catalog	2-11
message	2-11
messagebody	2-12
Simple Text Catalog Example	2-13
Elements of a Locale-Specific Catalog	2-14
locale_message_catalog	2-14
log_message	2-14
Other locale_message_catalog Elements	2-14
Locale Message Catalog Syntax	2-15

3. Using the BEA WebLogic Server Message Editor

About the Message Editor	3-1
Starting the Message Editor	3-3
Working with Catalogs	3-4
Browsing to an Existing Catalog	3-5
Creating a New Catalog	3-7
Adding Messages to Catalogs	3-9
Entering a New Log Message	3-9
Entering a New Simple Text Message	3-11
Finding Messages	3-12
Finding a Log Message	3-12
Finding a Simple Text Message	3-13
Using the Message Viewer	3-14
Viewing All Messages in a Catalog	3-14
Viewing All Messages in Several Catalogs	3-15
Choosing a Message to Edit from the Message Viewer	3-15
Editing an Existing Message	3-16

4. Using the BEA WebLogic Server Internationalization Utilities

About the WebLogic Server Internationalization Utilities	4-1
About the WebLogic Server Internationalization and Localization Interfaces	4-2
18ngen Utility	4-4
Syntax	4-4
Options	4-4
110ngen Utility	4-5

Syntax	4-5
Options.....	4-6
CatInfo Utility	4-6
Syntax	4-6
Options.....	4-7

A. Localizer Class Reference for BEA WebLogic Server

About Localizer Classes.....	A-1
Localizer Methods.....	A-2
Localizer Lookup Class.....	A-3

B. Logger Class Reference for BEA WebLogic Server

About Logger Classes	B-1
Example of a Generated Logger Class.....	B-2

C. Loggable Object Reference for BEA WebLogic Server

About Loggable Objects.....	C-1
How Loggable Objects Are Used.....	C-2

D. TextFormatter Class Reference for BEA WebLogic Server

About TextFormatter Classes.....	D-1
Example of an Application Using a TextFormatter Class.....	D-2



About This Document

This document defines internationalization and localization, and explains how to use the templates and tools provided with WebLogic Server to create or edit message catalogs that are locale-specific.

The document is organized as follows:

- [Chapter 1, “Overview of Internationalization for WebLogic Server,”](#) summarizes the processes required for internationalization and localization.
- [Chapter 2, “Using Message Catalogs with BEA WebLogic Server,”](#) describes message catalog types, message definitions, elements, and arguments.
- [Chapter 3, “Using the BEA WebLogic Server Message Editor,”](#) explains how to use the Message Editor that is included with WebLogic Server.
- [Chapter 4, “Using the BEA WebLogic Server Internationalization Utilities,”](#) explains how to use the internationalization utilities included with WebLogic Server.
- [Appendix A, “Localizer Class Reference for BEA WebLogic Server,”](#) describes `Localizer` classes, `Localizer` methods, key values for `Localizers`, and lookup properties for `Localizers`.
- [Appendix B, “Logger Class Reference for BEA WebLogic Server,”](#) describes `Logger` classes and provides an example of a message catalog and its corresponding `Logger` class.
- [Appendix C, “Loggable Object Reference for BEA WebLogic Server,”](#) describes `Loggable` objects and how they are used.
- [Appendix D, “TextFormatter Class Reference for BEA WebLogic Server,”](#) provides an example of an application that uses a `TextFormatter` class.

Audience

This document is written for application developers who must internationalize or localize the message catalogs included in the WebLogic Server distribution for locale-specific administration and management. It is assumed that readers are familiar with the WebLogic Server Platform and 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

For more information in general about internationalization and localization, refer to the following sources:

- The Java Developer Connection™ at java.sun.com
- The Internationalization section of the World Wide Web Consortium (W3C) Web Site at <http://www.w3.org>

Contact Us!

Your feedback on BEA documentation is important to us. Send us e-mail at 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.

Convention	Usage
Ctrl+Tab	Keys you press simultaneously.
<i>italics</i>	Emphasis and book titles.
monospace text	Code samples, commands and their options, Java classes, data types, directories, and file names and their extensions. Monospace text also indicates text that you enter from the keyboard. <i>Examples:</i> <pre>import java.util.Enumeration; chmod u+w * config/examples/applications .java config.xml float</pre>
<i>monospace italic text</i>	Variables in code. <i>Example:</i> <pre>String CustomerName;</pre>
UPPERCASE TEXT	Device names, environment variables, and logical operators. <i>Examples:</i> <pre>LPT1 BEA_HOME OR</pre>
{ }	A set of choices in a syntax line.
[]	Optional items in a syntax line. <i>Example:</i> <pre>java utils.MulticastTest -n name -a address [-p portnumber] [-t timeout] [-s send]</pre>

Convention	Usage
-------------------	--------------

	Separates mutually exclusive choices in a syntax line. <i>Example:</i> <pre>java weblogic.deploy [list deploy undeploy update] password {application} {source}</pre>
--	---

...	Indicates one of the following in a command line: <ul style="list-style-type: none">■ An argument can be repeated several times in the command line.■ The statement omits additional optional arguments.■ You can enter additional parameters, values, or other information
-----	---



1 Overview of Internationalization for WebLogic Server

The following sections provide an overview of localization and internationalization:

- [About Internationalization and Localization Standards](#)
- [Understanding Localization for WebLogic Server](#)
- [Understanding Message Catalogs](#)
- [Understanding Java Interfaces for Internationalization](#)
- [Main Steps for Creating an Internationalized Message](#)

About Internationalization and Localization Standards

BEA has adopted the World Wide Web Consortium's (W3C) recommendations for standard formats and protocols that are usable worldwide in all languages and in all writing systems. These standards are part of the Java internationalization Application Program Interfaces (APIs) that are used by WebLogic Server. *Internationalization*

(I18N) refers to the preparation of software so that it behaves properly in multiple locations. *Localization* (L10N) is the use of locale-specific language and constructs at run time.

Internationalization of textual data in WebLogic Server is provided through message catalogs. WebLogic Server supports message catalogs for log messages as well as simple text. Log messages contain data that is written to the log file. This data is predominantly dynamic and contains information that is specific to the current state of the application and system. When merged with text in a localized log message catalog, this data results in well-formatted, localized messages that describe the error condition in the language of the user. The output sent to the console is simple text. As with log messages, simple text can be merged with dynamic data.

The information covered in this guide addresses only the internationalization of textual data.

Understanding Localization for WebLogic Server

Localization covers not only language, but collation, date and time formats, monetary formats, and character encoding. Messages that are logged to the WebLogic Server error log can be localized to meet your particular requirements.

WebLogic Server internationalization supports localization of two types of data:

- **Log messages**—Log messages are informational messages that are written to the server log, and may also contain error messages if the appropriate message arguments are included in the message definition.
- **Simple text**—Simple text is any text other than log messages and exceptions that the server must display, such as the output from a utility. Examples of simple text include usage messages, graphical user interface (GUI) labels, and error messages.

Understanding Message Catalogs

All internationalized text is defined in message catalogs, each of which defines a collection of log messages or simple text. To create an internationalized message, you externalize all message strings in a message catalog so that the strings can be easily converted to multiple locales without changing or recompiling the code. The application code supplies run-time values to the logging methods. The logging methods merge the code with the message string in the catalog per the current locale. And the application code then prints a localized message in the log files.

There are three types of message catalogs:

- **Log message catalogs**—Collections of log messages.
- **Simple text catalogs**—Collections of simple text messages.
- **Locale message catalogs**—Collections of locale-specific messages corresponding to a top-level log message or simple text catalog.

Message IDs in log message catalogs or locale message catalogs are unique across all log message or locale message catalogs. Within the message catalog file, each localized version of the message is assigned a unique message ID and message text specific to the error. Ideally, a message is logged from only one location within the system so that a support team can easily find it. Message IDs in simple text catalogs are unique within each simple text catalog.

Refer to [“Using Message Catalogs with BEA WebLogic Server”](#) on page 2-1 for more detailed information about message catalogs.

Understanding Java Interfaces for Internationalization

WebLogic Server uses the Java internationalization interfaces for providing internationalization and localization. In addition to understanding how WebLogic Server handles internationalization, users should be familiar with the Java internationalization interfaces and the following classes included in the Java Development Kit (JDK):

Class	Description
<code>java.util.Locale</code>	Represents a specific geographical, political, or cultural region.
<code>java.util.ResourceBundle</code>	Provides containers for locale-specific objects.
<code>java.text.MessageFormat</code>	Produces concatenated messages in a language-neutral way.

Main Steps for Creating an Internationalized Message

The following steps describe how you create an internationalized message for use with WebLogic Server:

1. Create or edit a top-level log catalog or simple text catalog by defining the messages in the catalog. For details, see [“Using the BEA WebLogic Server Message Editor” on page 3-1](#).

In addition to message text, include information about the type and placement of any run-time values that the message contains.

Main Steps for Creating an Internationalized Message

2. Run `i18ngen` to validate the catalog you created or edited in Step 1 and generate runtime classes.

The generated classes contain a method for each message. The method is defined according to information specified in the message catalog entry. The methods include `Logger` or `TextFormatter` methods, depending on the type of catalog. For details, see [“i18ngen Utility” on page 4-4](#).

3. Create locale-specific catalogs as required for the message catalog you created in Step 1.
4. Run `i18ngen` to process the locale-specific catalogs.
5. Instrument the application to use the `Logger` or `TextFormatter` methods you generated in Step 2. When the application logs or returns a message, the message is written using the localized version of the text according to the `Logger` or `TextFormatter` methods used.

These steps are described in detail in the following topics. For more detailed information, including an overview of the logging subsystem and a description of log message parts, see [“Using Log Messages to Manage WebLogic Servers”](#) in the *WebLogic Server Administration Guide*.

1 *Overview of Internationalization for WebLogic Server*

2 Using Message Catalogs with BEA WebLogic Server

The following sections describe message catalogs and how to use them:

- [Overview of Message Catalogs](#)
- [Message Catalog Hierarchy](#)
- [Choosing Names for Message Catalogs](#)
- [Using Message Arguments](#)
- [Message Catalog Formats](#)

Overview of Message Catalogs

Message catalogs are XML files that contain a description of a collection of text messages, each indexed by a unique identifier. You compile these XML files into classes during the `i18ngen` utility build process. (Refer to “[18ngen Utility](#)” on [page 4-4](#) for more information). The methods of the resulting classes are the objects used to log messages at runtime.

Message catalogs support multiple locales or languages. For a specific message catalog there is exactly one default version, known as the top-level catalog. Then there are corresponding locale-specific catalogs, one for each additional supported locale. The top-level catalog includes all the information necessary to define the message. The locale-specific catalogs contain only the message ID, the date changed, and the translation of the message for the specific locale.

The message catalog files are defined by an XML document type definition (DTD). The DTDs are stored in the `msgcat` directory. The location of the `msgcat` directory may vary, depending upon where you installed WebLogic Server. If you used the default installation path, the `msgcat` directory is located in the `BEA_HOME` directory under `WL_HOME\samples\server\src\examples\i18n\msgcat`.

Two DTDs are included in the WebLogic Server distribution:

- `msgcat.dtd`—Describes the syntax of top-level, default catalogs.
- `i18n_msgcat.dtd`—Describes the syntax of locale-specific catalogs.

The `msgcat` directory also contains templates that you can use to create top-level and locale-specific message catalogs.

Users may choose to create a single log message catalog for all their logging requirements, or create smaller catalogs based on a subsystem or Java package. We recommend using multiple subsystems because you can focus on specific portions of the log during viewing.

For simple text catalogs, the recommended approach is to create a single catalog for each utility being internationalized. Developers can create site-specific message catalogs using the Message Editor as described in [“Using the BEA WebLogic Server Message Editor” on page 3-1](#).

Message Catalog Hierarchy

All messages and exceptions must be defined in the default, top-level catalog. The WebLogic Server distribution includes a collection of sample catalogs in the `WL_HOME\samples\server\src\examples\i18n\msgcat` directory.

Note: This directory path may vary, depending on where you chose to install WebLogic Server.

Catalogs that provide different localizations of the base catalogs are defined in `msgcat` subdirectories named for the locale (for example, `msgcat\de` for Germany). You might have a top-level catalog named `mycat.xml`, and a German translation of it called `..\de\mycat.xml`. Typically the top-level catalog is English, but English is not required for any catalogs except the installed WebLogic Server catalogs.

Locale designations (for example, `de`) also have a hierarchy as defined in the `java.util.Locale` documentation. A locale can include a language, country, and variant. Language is the most common locale designation. Language can be extended with a country code. For instance, `en\US`, indicates American English. The name of the associated catalog is `..\en\US\mycat.xml`. Variants are vendor or browser-specific and are used to introduce minor differences (for example, collation sequences) between two or more locales defined by either language or country.

Choosing Names for Message Catalogs

Because the name of a message catalog file (without the `.xml` extension) is used to generate runtime class and property names, you should choose the name carefully.

Follow these guidelines for naming message catalogs:

- Do not choose a message catalog name that conflicts with any names of existing classes in the target package.
- The message catalog name should only contain characters that are allowed in class names.
- Follow class naming standards.

For example, the resulting class names for a catalog named `xyz.xml` are `xyzLogLocalizer` and `xyzLogger`.

The following considerations also apply to message catalog files:

- Message IDs are generally six-character strings with leading zeros. Some interfaces also support integer representations.

- Java allows you to group classes into a collection called a *package*. A package name should be consistent with the name of the subsystem in which a particular catalog resides.
- The log Localizer “classes” are actually `ResourceBundle` property files.

Using Message Arguments

The message body, message detail, cause, and action sections of a log message can include message arguments, as described by `java.text.MessageFormat`. Only the message body section in a simple message can include arguments. Arguments are values that can be dynamically set at runtime. These values are passed to routines, such as printing out a message. A message can support up to 10 arguments, numbered 0-9. You can include any subset of these arguments in any text section of the message definition, although the message body should include all of the arguments. You insert message arguments into a message definition during development, and these arguments are replaced by the appropriate message content at runtime when the message is logged.

The following excerpt from an XML log message definition shows how you can use message arguments. The argument number must correspond to one of the arguments specified in the `method` attribute. Specifically, `{0}` with the first argument, `{1}` with the second, and so on.

Listing 2-1 Example of Message Arguments

```
<messagebody>Unable to open file, {0}</messagebody>
  <messagedetail>
    File, {0} does not exist. The server will restore the file
    contents from {1}, resulting in the use of default values
    for all future requests.
  </messagedetail>
  <cause>The file was deleted</cause>
  <action>
    If this error repeats then investigate unauthorized access to
    the file system.
  </action>
```

An example of a `method` attribute for the above message is as follows:

```
-method="logNoFile(String name, String path)"
```

The message expects two arguments, `{0}` and `{1}`:

- `{0}` is used in the message body.
- Both are used in the message detail.
- Neither is used in the `<cause>` or `<action>` section.

In addition, the arguments are expected to be strings, or representable as strings. Numeric data is represented as `{n,number}`. Dates are supported as `{n,date}`. You must assign a severity level for log messages. Log messages are generated through the generated `Logger` methods, as defined by the `method` attribute.

Message Catalog Formats

The catalog format for top-level and locale-specific catalog files is slightly different. The top-level catalogs define the textual messages for the base locale. Locale-specific catalogs only provide translations of text defined in the top-level version. Log message catalogs are defined differently from simple text catalogs.

Elements of each of these types of catalogs are described in the following sections:

- [Elements of a Log Message Catalog](#)
- [Elements of a Simple Text Message Catalog](#)
- [Elements of a Locale-Specific Catalog](#)

Elements of a Log Message Catalog

This section provides reference information for the following elements of a log message catalog:

- [message_catalog](#)

- [log_message](#)
- [Other log_message Catalog Elements](#)

message_catalog

The following table describes the attributes that you can define for the `message_catalog` element.

Attribute	Default	Required/ Optional	Description
<code>i18n_package</code>	<code>weblogic.i18n</code>	Optional	Java package containing generated Logger classes for this catalog. The classes are named after the catalog file name. For example, for a catalog using <code>mycat.xml</code> , a generated logger class called <code>i18n_package.mycatLogger.class</code> .
<code>i10n_package</code>	<code>weblogic.i18n</code>	Optional	Java package containing generated <code>LogLocalizer</code> properties for this catalog. Classes are named after the catalog file name. For example, for a catalog called <code>mycat.xml</code> a properties file called <code>i10n_package.mycatLogLocalizer.properties</code> is generated.
<code>subsystem</code>	None	Required	An acronym identifying the subsystem associated with this catalog. The name of the subsystem is included in the error log and is used for message isolation purposes.
<code>version</code>	None	Required	Specifies the version of the <code>msgcat.dtd</code> being used. The format is <code>n.n</code> , for example, <code>version="1.0"</code> . Must be at least "1.0".
<code>baseid</code>	000000 for WebLogic Server catalogs 500000 for user-defined catalogs	Optional	Specifies the lowest message ID used in this catalog. The syntax is one to six decimal units.

Attribute	Default	Required/ Optional	Description
endid	499999 for WebLogic Server catalogs 999999 for user-defined catalogs	Optional	Specifies the highest message ID used in this catalog. The syntax is one to six decimal units.

log_message

The following table describes the attributes that you can define for the `log_message` element.

Attribute	Default	Required/ Optional	Description
messageid	None	Required	Unique identifier for this log message. Uniqueness should extend across all catalogs. Value must be in range defined by baseid and endid attributes. This is a child element of <code>message_catalog</code> .
datelastchanged	None	Optional	Date/time stamp used for managing modifications to this message. The date is supplied by utilities that run on the catalogs. The syntax is: <code>Long.toString(new Date().getTime());</code>
severity	None	Required	Indicates the severity of the log message. Must be one of the following: <code>debug</code> , <code>info</code> , <code>warning</code> , <code>notice</code> , <code>error</code> , <code>critical</code> , <code>alert</code> , or <code>emergency</code> . User catalogs may only use <code>debug</code> , <code>info</code> , <code>warning</code> , and <code>error</code> .
stacktrace	true	Optional	Indicates whether to generate a stack trace for Throwable arguments. Possible values are <code>true</code> or <code>false</code> . When the value is <code>true</code> a trace is generated. The syntax is: <code>stacktrace="true"</code>

2 Using Message Catalogs with BEA WebLogic Server

Attribute	Default	Required/ Optional	Description
method	None	Required	<p>Method signature for logging this message. Two methods are actually provided: the one specified here and a similar one with an additional Throwable argument.</p> <p>The syntax is the standard Java method signature, without the qualifiers, semicolon, and extensions. Argument types can be any Java primitive or class. Classes must be fully qualified if not in <code>java.lang</code>. Classes must also conform to <code>java.text.MessageFormat</code> conventions. In general, class arguments should have a useful <code>toString()</code> method.</p> <p>Arguments can be any valid name, but should follow the convention of <code>argn</code> where <code>n</code> is 0 thru 9. There can be no more than ten (10) arguments. For each <code>argn</code> there should be at least one corresponding placeholder in the text elements described in “Other log_message Catalog Elements” on page 2-8. Placeholders are of the form <code>{n}</code>, <code>{n, number}</code> or <code>{n, date}</code>.</p>
loggables	False	Optional	<p>Indicates whether to generate methods to return Loggable objects for each message. The syntax is: <code>loggables="true"</code></p> <p>Refer to “Loggable Object Reference for BEA WebLogic Server” on page C-1</p>

Other log_message Catalog Elements

The following table describes the child elements of the `log_message` element.

Element	Parent Element	Required/Optional	Description
messagebody	log_message	Required	A string containing a short description for this message. This element may contain zero or more placeholders, {n} that are replaced by the appropriate argument when the log message is localized.
messagedetail	log_message	Required	A string containing a detailed description of the event. This element may contain zero or more placeholders, {n} that are replaced by the appropriate argument when the log message is localized.
cause	log_message	Required	A string describing the root cause of the problem. This element may contain zero or more placeholders, {n} that are replaced by the appropriate argument when the log message is localized.
action	log_message	Required	A string describing the recommended resolution. This element may contain zero or more placeholders, {n} that are replaced by the appropriate argument when the log message is localized.

Log Message Catalog Example

The following example shows a log message catalog, `MyUtilLog.xml`, with one log message.

Listing 2-2 Example of a Log Message Catalog

```
<?xml version="1.0"?>
<!DOCTYPE message_catalog PUBLIC "weblogic-message-catalog-dtd"
"http://www.bea.com/servers/wls700/dtd/msgcat.dtd">
<message_catalog
  l10n_package="programs.utils"
  i18n_package="programs.utils"
  subsystem="MYUTIL">
```

```
version="1.0"
baseid="600000"
endid="600100"
<log_message
  messageid="600001"
  severity="warning"
  method="logNoAuthorization(String arg0, java.util.Date arg1,
    int arg2)"
  <messagebody>
    Could not open file, {0} on {1,date} after {2,number} attempts.
  </messagebody>
  <messagedetail>
    The configuration for this application will be defaulted to
    factory settings. Custom configuration information resides
    in file, {0}, created on {1,date}, but is not readable.
  </messagedetail>
  <cause>
    The user is not authorized to use custom configurations. Custom
    configuration information resides in file, {0}, created on
    {1,date}, but is not readable.The attempt has been logged to
    the security log.
  </cause>
  <action>
    The user needs to gain appropriate authorization or learn to
    live with the default settings.
  </action>
</log_message>
</message_catalog>
```

Elements of a Simple Text Message Catalog

This section provides reference information for the following simple text message catalog elements:

- [message_catalog](#)
- [message](#)
- [messagebody](#)

message_catalog

The following table describes the attributes that you can define for the `message_catalog` element.

Attribute	Default	Required/ Optional	Description
<code>l10n_package</code>	<code>weblogic.i18n</code>	Optional	Java package containing generated <code>TextFormatter</code> classes and <code>TextLocalizer</code> properties for this catalog. Classes are named after the catalog file name. For example, for a catalog called <code>mycat.xml</code> , a properties file called <code>l10n_package.mycatTextLocalizer</code> properties is generated.
<code>subsystem</code>	None	Required	An acronym identifying the subsystem associated with this catalog. The name of the subsystem is included in the error log and is used for message isolation purposes.
<code>version</code>	None	Required	Specifies the version of the <code>msgcat.dtd</code> being used. The format is <code>n.n</code> , for example, <code>version="1.0"</code> . Must be at least "1.0".

message

The following table describes the attributes that you can define for the `message` element.

Attribute	Default	Required/ Optional	Description
<code>messageid</code>	None	Required	Unique identifier for this log message in alpha-numeric string format. Uniqueness is required only within the context of this catalog. <code>message</code> is a child element of <code>message_catalog</code> .
<code>datelastchanged</code>	None	Optional	Date/time stamp useful for managing modifications to this message.

Attribute	Default	Required/ Optional	Description
method	None	Optional	<p>Method signature for formatting this message.</p> <p>The syntax is a standard Java method signature, less return type, qualifiers, semicolon, and extensions. The return type is always <code>String</code>. Argument types can be any Java primitive or class. Classes must be fully qualified if not in <code>java.lang</code>. Classes must also conform to <code>java.text.MessageFormat</code> conventions. In general, class arguments should have a useful <code>toString()</code> method, and the corresponding <code>MessageFormat</code> placeholders must be strings; they must be of the form <code>{n}</code>. Argument names can be any valid name. There can be no more than 10 arguments.</p> <p>For each argument there must be at least one corresponding placeholder in the <code>messagebody</code> element described below. Placeholders are of the form <code>{n}</code>, <code>{n,number}</code> or <code>{n,date}</code>.</p> <p>Example:</p> <pre>method="getNoAuthorization (String filename, java.util.Date creDate) "</pre> <p>This example would result in a method in the <code>TextFormatter</code> class as follows:</p> <pre>public String getNoAuthorization (String filename, java.util.Date creDate)</pre>

messagebody

The following table describes the child element of the message element.

Element	Parent Element	Required/Optional	Description
messagebody	message	Required	The text associated with the message. This element may contain zero or more placeholders {n} that will be replaced by the appropriate arguments when the log message is localized.

Simple Text Catalog Example

The following example shows a simple text catalog, `MyUtilLabels.xml`, with one text definition.

Listing 2-3 Example of a Simple Text Catalog

```
<?xml version="1.0"?>
<!DOCTYPE message_catalog PUBLIC "weblogic-message-catalog-dtd"
    "http://www.bea.com/servers/wls700/dtd/msgcat.dtd">
<message_catalog>
  l10n_package="programs.utils"
  i18n_package="programs.utils"
  subsystem="MYUTIL"
  version="1.0"
  <message>
    messageid="FileMenuTitle"
    <messagebody>
      File
    </messagebody>
  </message>
</message_catalog>
```

Elements of a Locale-Specific Catalog

The locale-specific catalogs are subsets of top-level catalogs. They are maintained in subdirectories named for the locales they represent. The elements and attributes described in the following sections are valid for locale-specific catalogs.

locale_message_catalog

The following table describes the attributes that you can define for the `locale_message_catalog` element.

Attribute	Default	Required/ Optional	Description
<code>l10n_package</code>	<code>weblogic.i18n</code>	Optional	Java package containing generated <code>LogLocalizer</code> or <code>TextLocalizer</code> properties for this catalog. properties are named after the catalog file name. For example, for a French log message catalog called <code>mycat.xml</code> , a properties file called <code>l10n_package.mycatLogLocalizer_fr_FR.properties</code> properties is generated.
<code>version</code>	None	Required	Specifies the version of the <code>msgcat.dtd</code> being used. The format is <code>n.n</code> , for example, <code>version="1.0"</code> . Must be at least "1.0".

log_message

The locale-specific catalog uses the attributes defined for the `log_message` element in the log message catalog so this element does not need to be defined.

Other locale_message_catalog Elements

The locale-specific catalog uses the `messagebody`, `messagedetail`, `cause`, and `action` catalog elements defined for the log message catalog so these elements do not need to be defined.

Locale Message Catalog Syntax

The following example shows a French translation of a message that is available in `...\msgcat\fr\MyUtilLabels.xml`.

The translated message appears as shown in [Listing 2-4](#).

Listing 2-4 Example of a Message Translated to French

```
<?xml version="1.0"?>
<!DOCTYPE message_catalog PUBLIC
    "weblogic-locale-message-catalog-dtd"
    "http://www.bea.com/servers/wls700/dtd/l10n_msgcat.dtd">
<locale_message_catalog
    l10n_package="programs.utils"
    subsystem="MYUTIL"
    version="1.0">
  <message>
    <messageid="FileMenuTitle">
      <messagebody> Fichier </messagebody>
    </message>
  </locale_message_catalog>
```

When entering text in the `messagebody`, `messagedetail`, `cause` and `action` elements, you must use a tool that generates valid Unicode Transformation Format-8 (UTF-8) characters, and have appropriate keyboard mappings installed. UTF-8 is an efficient encoding of Unicode character-strings that optimizes the encoding ASCII characters. Message catalogs always use UTF-8 encoding. The `MessageLocalizer` utility that is downloaded with WebLogic Server is a tool that can be used to generate valid UTF-8 characters.

3 Using the BEA WebLogic Server Message Editor

The following sections describe how to use the Message Editor:

- [About the Message Editor](#)
- [Starting the Message Editor](#)
- [Working with Catalogs](#)
- [Adding Messages to Catalogs](#)
- [Finding Messages](#)
- [Using the Message Viewer](#)
- [Editing an Existing Message](#)

About the Message Editor

The Message Editor is a graphical interface tool that allows you to create, read, and write XML message catalogs. The Message Editor is installed when you install WebLogic Server. Optionally, you can also edit the XML catalogs directly in a text editor.

Note: The Message Editor does not support the editing of localized catalogs.

The Message Editor allows you to perform the following tasks:

- Create XML message catalogs
- Create and edit messages
- View all the messages in one catalog
- View all the messages in several catalogs simultaneously
- Search for messages
- Validate the XML in catalog entries

The catalog currently being created or used in the Message Editor is the *context catalog*.

The catalog parsers (`i18ngen` or `l10ngen`) recognize character code references for special characters and convert them to the intended character, as shown in [Table 3-1](#). The Message Editor recognizes the special characters and writes them back using the character code references. The parsers recognize no other character references.

Table 3-1 Special Character Reference

Character Code	Special Character
<code>&amp;</code>	@
<code>&lt;</code>	<
<code>&gt;</code>	>
<code>&apos;</code>	'
<code>&quot;</code>	"

Starting the Message Editor

Before you start the Message Editor, you should have installed and configured your WebLogic Server system and set your environment variables (`setExamplesEnv.cmd`). Make sure that your classpath is set correctly.

Sample message catalog files are located in your `BEA_HOME\samples\wlserver7.0\samples\examples\i18n\msgcat` directory.

Note: This directory path may vary, depending on where you chose to install WebLogic Server.

To start the Message Editor, type:

```
java weblogic.MsgEditor
```

or

```
java weblogic.i18ntools.gui.MessageEditor
```

The main WebLogic Message Editor window for **Log Messages** displays as shown in [Figure 3-1](#).

Figure 3-1 WebLogic Message Editor for Log Messages



Working with Catalogs

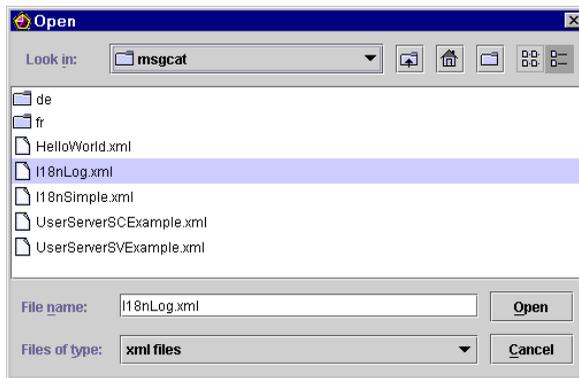
The following sections describe how to use the Message Editor to manage catalogs:

- [Browsing to an Existing Catalog](#)
- [Creating a New Catalog](#)

Browsing to an Existing Catalog

To find an existing catalog from the main WebLogic Message Editor window, enter the full pathname in the **Message Catalog** field, or click **Browse** and navigate to the existing catalog from the Open dialog.

Figure 3-2 Open



The sample catalogs included with your WebLogic Server installation are in the `BEA_HOME\samples\wlserver7.0\samples\examples\i18n\msgcat` directory.

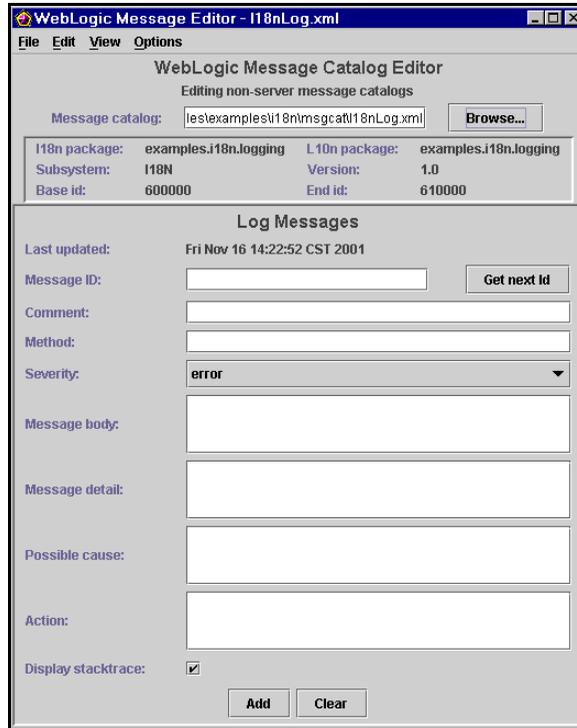
Note: This directory path may vary, depending on where you chose to install WebLogic Server.

Your user-defined catalogs may be located in any directory you designate.

Once the catalog has been located, the **Packages**, **Subsystem**, **Version**, **Base ID**, and **End ID** (if any) for that catalog are displayed, and that catalog is the context catalog in which all other actions are performed. You are now ready to enter new messages into that catalog, to edit existing messages, to search for a message, or to view all messages in the catalog.

If a log message catalog is selected in the **Message catalog** field, the WebLogic Message Editor window for **Log Messages** displays as shown in [Figure 3-3](#).

Figure 3-3 WebLogic Message Editor for Log Messages



If a simple messages catalog is selected in the **Message catalog** field, the WebLogic Message Editor window for **Simple Messages** displays as shown in [Figure 3-4](#).

Figure 3-4 WebLogic Message Editor for Simple Messages



Creating a New Catalog

To create a new catalog, complete the following procedure:

1. Choose the **File** menu from the main menu bar of the WebLogic Message Editor window.
2. Choose **New Catalog**.

The “Create new catalog” dialog displays as shown in [Figure 3-5](#).

Figure 3-5 Create New Catalog

The screenshot shows a dialog box titled "Create new catalog". It has a standard Windows-style title bar with a close button. The dialog contains the following fields and controls:

- Message catalog:** A text input field followed by a "Browse..." button.
- Catalog type:** A dropdown menu currently showing "Log messages".
- I18n package:** A text input field.
- L10n package:** A text input field.
- Subsystem:** A text input field.
- Version:** A text input field containing the value "1.0".
- Base id:** A text input field.
- End id:** A text input field.
- At the bottom, there are two buttons: "Create Catalog" and "Cancel".

3. In the **Message Catalog** field, enter the full pathname and the name of the new catalog, which must include the `xml` extension. Or, click **Browse** and navigate to the WebLogic catalog directory, `msgcat`.

4. Use the drop-down **Catalog type** list to indicate whether your catalog is to be a **Log message** catalog or a **Simple text** message catalog.

If you select a log message catalog, the **Base ID** and **End ID** fields are displayed. These fields indicate the range of ID numbers for messages in the catalog. If you select a simple text message catalog, these fields disappear.

5. Enter the name of the package where you want to place generated Logger classes in the **I18n package** field. The default is `weblogic.i18n`. If you want to place your logger classes in another package with your application, specify the package name here.

6. Enter the name of the package where you want to place the catalog data in the **L10n package** field. The default is `weblogic.i18n`. If you want to place your catalog data in another package with your application, specify the package name here.

7. Enter a name in the **Subsystem** field to indicate which part of the system logged the message. This name is logged with the message. For applications, the application name is normally entered in the **Subsystem** field.

8. Click **Create Catalog**.

The "Create new catalog" dialog closes, and the catalog you just created is displayed as the context catalog in the main Message Editor window.

Adding Messages to Catalogs

The following sections describe how to use the Message Editor to add messages to catalogs:

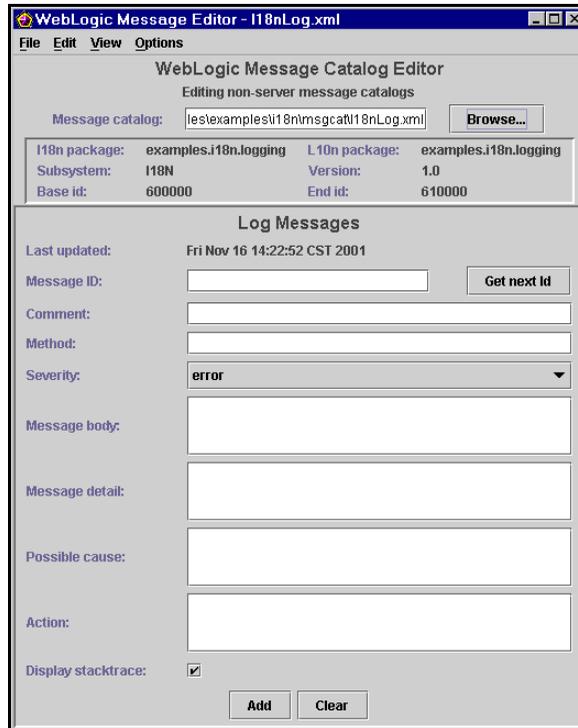
- [Entering a New Log Message](#)
- [Entering a New Simple Text Message](#)

Entering a New Log Message

To enter a new message into a log catalog, complete the following procedure:

1. From the WebLogic Message Editor main dialog as shown in [Figure 3-6](#), enter the full pathname in the **Message Catalog** field or click **Browse** and navigate to the existing catalog.

Figure 3-6 Log Messages



2. Enter a unique alphanumeric **Message ID** or click **Get next ID** to get the next unique numerical ID in the context catalog.
3. Enter the appropriate **Method** for your log message, including parentheses and any arguments. For example:

```
logNoAuthorization(String arg0, java.util.Date arg1,  
int arg2) logNote()
```

4. Choose a **Severity** from the list of possible levels.
5. Enter text for the **Message body**, **Message detail**, **Possible cause**, and **Action**. Parameters are denoted by {*n*}. For example:

```
User {0} tried to access this on {1} but has no authority to do  
so. {2} lashes with a keyboard with coke spilled on it.
```

6. Toggle the **Display stacktrace** option on or off by clicking the checkmark box. Use this option to print a stacktrace along with the message when a Logger method takes an exception as one of its arguments.
7. Click **Add**.
The message is added and the entire catalog is immediately written to disk.

Entering a New Simple Text Message

To enter a simple text message into a simple messages catalog, complete the following procedure:

1. From the WebLogic Message Editor main dialog, enter the full pathname in the **Message Catalog** field or click **Browse** and navigate to the existing catalog.

The WebLogic Message Editor for Simple Messages dialog displays as shown in [Figure 3-7](#).

Figure 3-7 Simple Messages



2. Enter a unique alphanumeric **Message ID**.
3. Enter a **Comment** if required.
4. Enter the appropriate **Method** for your simple message, including parentheses and any arguments.

5. Enter the **Message body** text.
6. Click **Add**.

The message is added and the entire catalog is immediately written to disk.

Finding Messages

The following sections describe how to use the Message Editor to find messages:

- [Finding a Log Message](#)
- [Finding a Simple Text Message](#)

Finding a Log Message

To find a log message, complete the following procedure:

1. Make sure that the context catalog is a log message catalog and the WebLogic Message Editor **Log Messages** main window is displayed as shown in [Figure 3-3](#).
2. Choose **Edit** from the main menu bar.
3. Choose **Search** to display the “Search for Log Message” dialog as shown in [Figure 3-8](#).

Figure 3-8 Search for Log Message



4. Enter the **Message ID** and the **Method** name.

5. Enter as much information as you need in the **Message text search** field to find the correct message. The search for text does a partial match in any of the text fields.
6. Click **Find first** or **Find next**.
The fields are strung together to find the message. If a matching message is found, it displays in the main Message Editor window as shown in [Figure 3-1](#).

Finding a Simple Text Message

To find a simple text message, complete the following procedure:

1. Make sure that the context catalog is a simple text message catalog and the WebLogic Message Editor **Simple Messages** main window is displayed as shown in [Figure 3-4](#).
2. Choose **Edit** from the main menu bar.
3. Choose **Search** to display the “Search for Simple Message” dialog as shown in [Figure 3-9](#).

Figure 3-9 Search for Simple Message



4. Enter the **Message ID**.
5. Enter as much information as you need in the **Message text search** field to find the correct message. The search for text does a partial match in any of the text fields.
6. Click **Find first** or **Find next**.
The fields are strung together to find the message. If a matching message is found, it displays in the main Message Editor window as shown in [Figure 3-4](#).

Using the Message Viewer

The WebLogic Message Editor contains a Message Viewer that allows you to view all messages in a catalog, view all messages in multiple catalogs, and choose any message to edit.

The following sections describe how to use the Message Viewer to view and choose messages to edit:

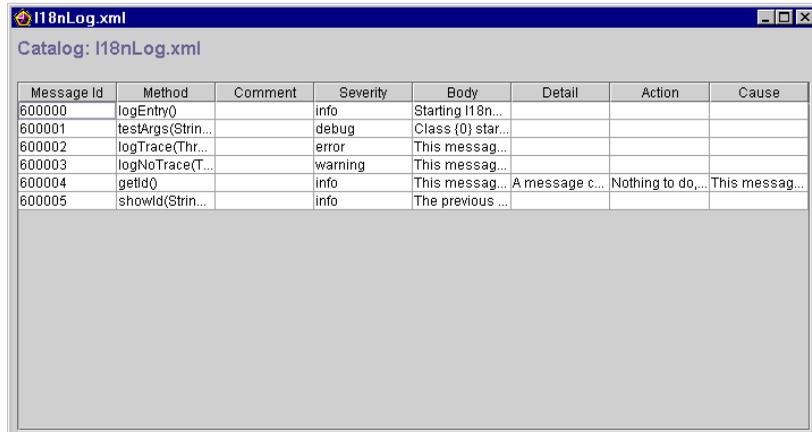
- [Viewing All Messages in a Catalog](#)
- [Viewing All Messages in Several Catalogs](#)
- [Choosing a Message to Edit from the Message Viewer](#)

Viewing All Messages in a Catalog

To view all the messages in a catalog:

1. Open the WebLogic Message Editor. The main WebLogic Message Editor window displays with the catalog for the last message viewed as the current context catalog.
2. Choose **View** from the menu bar. All the messages for the current context catalog display in a table in the Message Viewer window, as shown in [Figure 3-10](#). The Message Viewer displays in a separate window from the Message Editor and the Message Editor remains open.

Figure 3-10 Message Viewer



The screenshot shows a window titled 'I18nLog.xml' with a sub-header 'Catalog: I18nLog.xml'. Below the header is a table with the following data:

Message Id	Method	Comment	Severity	Body	Detail	Action	Cause
600000	logEntry()		info	Starting I18n...			
600001	testArgs(Strin...		debug	Class {0} star...			
600002	logTrace(Thr...		error	This messag...			
600003	logNoTrace(T...		warning	This messag...			
600004	getId()		info	This messag...	A message c...	Nothing to do,...	This messag...
600005	showId(Strin...		info	The previous ...			

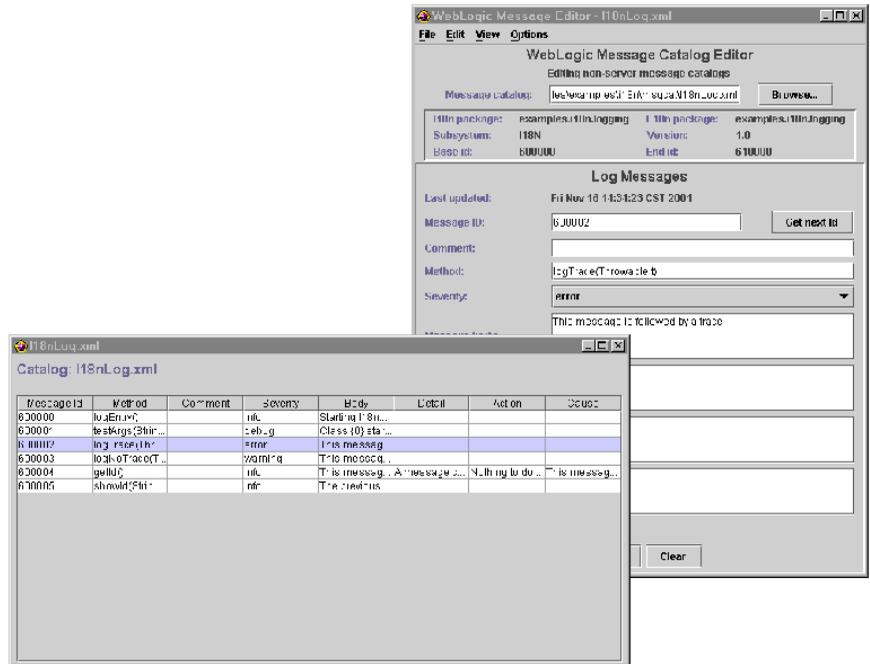
Viewing All Messages in Several Catalogs

If you view the messages from the current context catalog, and then change the context by clicking **Browse** on the WebLogic Message Editor main window to navigate to a new catalog, the old view of the old catalog remains on the screen while you view the new catalog in a second Message Viewer window. Repeating this step allows you to view messages for as many catalogs as you require (or can reasonably fit on your screen). Each catalog displays in a separate Message Viewer window. Refer to [“Browsing to an Existing Catalog” on page 3-5](#) for information about Browsing to a new catalog.

Choosing a Message to Edit from the Message Viewer

After you use the Message Viewer to view a list of messages, you can click on any message displayed in any row of the Message Viewer. The catalog for the selected message becomes the context catalog and the message displays in the Message Editor main window.

Figure 3-11 Message Viewer and Message Editor for Message 600002



Editing an Existing Message

To edit an existing message, complete the following procedure:

1. Find the message you want to edit, either by using the Search dialog as described in [Finding a Log Message](#) and [Finding a Simple Text Message](#), or by clicking on a row in the message viewer as described in [Choosing a Message to Edit from the Message Viewer](#).
2. Edit the fields you wish to change in the main Message Editor window.
3. Click **Update**.

The message is updated and the entire catalog is immediately written to disk.

4 Using the BEA WebLogic Server Internationalization Utilities

The following sections contain information about the WebLogic Server utilities used for internationalization and localization:

- [About the WebLogic Server Internationalization Utilities](#)
- [About the WebLogic Server Internationalization and Localization Interfaces](#)
- [l8ngen Utility](#)
- [l10ngen Utility](#)
- [CatInfo Utility](#)

About the WebLogic Server Internationalization Utilities

WebLogic Server provides three internationalization utilities:

- [i18ngen Utility](#)—Message catalog parser. Use this utility to generate classes used for localizing text in log messages.
- [i10ngen Utility](#)—Locale-specific message catalog parser. Use this utility to process locale-specific catalogs.
- [CatInfo Utility](#)—Utility that lists installed log messages. Use this utility to generate a list of installed log messages.

Note: Text in the catalog definitions may contain formatting characters for readability (for example, end of line characters), but these are not preserved by the parsers. Text data is normalized into a one-line string. All leading and trailing white space is removed. All embedded end of line characters are replaced by spaces as required to preserve word separation. Tabs are left intact.

About the WebLogic Server Internationalization and Localization Interfaces

The `i18ngen` utility validates message catalogs, creating the necessary runtime classes for producing localized messages. The `i10ngen` utility validates locale-specific catalogs, creating additional properties files for the different locales defined by the catalogs.

You can internationalize simple text-based utilities that you are running on WebLogic Server by specifying that those utilities must use `Localizers` to access text data. You instrument the applications with `Logger` and `TextFormatter` classes generated from the `i18ngen` utility. Refer to “[18ngen Utility](#)” on page 4-4 for detailed information about the `i18ngen` utility.

The generated `Logger` classes are used for logging purposes instead of the traditional method of writing English text to a log. For example, `i18ngen` generates a class `xyzLogger` in the appropriate package for the catalog `xyz.xml`.

As another example, when the `MyUtilLog.xml` catalog is used, the class `programs.utils.MyUtilLogger.class` is generated. For each log message defined in the catalog, this class contains static public methods as defined by the `method` attribute.

`TextFormatter` classes are generated for each simple message catalog. These classes include static methods for accessing localized and formatted text from the catalog. They are convenience classes that handle the interface with the message body, placeholders, and `MessageFormat`. You specify the formatting methods through the `method` attribute in each message definition. For example, if the definition of a message in a catalog includes the attribute `method=getErrorNumber(int err)`, the `TextFormatter` class shown in [Listing 4-1](#) is generated.

Listing 4-1 Example of a TextFormatter Class

```
package my.text;
public class xyzTextFormatter
{
    . . .
    public String getErrorNumber(int err)
    {
        . . .
    }
}
```

[Listing 4-2](#) shows an example of how the `getErrorNumber` method could be used in code.

Listing 4-2 Example of getErrorNumber Method

```
import my.text.xyzTextFormatter
. . .

xyzTextFormatter xyzL10n = new xyzTextFormatter();
System.out.println(xyzL10n.getErrorNumber(someVal));
```

The output prints the message text in the current locale, with the `someVal` argument inserted appropriately.

18ngen Utility

The `i18ngen` utility parses message catalogs (XML files) to produce `Logger` and `TextFormatter` classes used for localizing the text in log messages. The top-level message catalogs are compiled into Java classes and properties files. The `i18ngen` utility also creates or updates the `i18n_user.properties` file. Any errors, warnings, or informational messages are sent to `stderr`.

Syntax

```
java weblogic.i18ngen [options] files
```

Note: Utilities can be run from any directory, but if files are listed on the command line, then their path is relative to the current directory.

Options

Option	Definition
<code>-d targetdirectory</code>	Specifies the root directory to which generated Java source files are targeted. User catalog properties are placed in <code>i18n_user.properties</code> , relative to the designated target directory. Files are placed in appropriate directories based on the <code>i18n_package</code> and <code>l10n_package</code> values in the corresponding message catalog. The default target directory is the current directory.
<code>-n</code>	Parse and validate, but do not generate classes
<code>-keepgenerated</code>	Keep generated Java source
<code>-ignore</code>	Ignore errors

<code>-i18n</code>	Creates the internationalizer source (for example, <code>*Logger.java</code>) that supports the logging of internationalized messages.
<code>-l10n</code>	Creates the localizer source (property resource bundles) that provides access to each message defined in each message catalog. These properties are used by localization utilities to localize messages.
<code>-compile</code>	Compiles generated Java files using the current CLASSPATH. The resulting classes are placed in the directory identified by the <code>-d</code> option. Errors detected during compilation generally result in no class files or properties file being created. <code>i18ngen</code> exits with a bad exit status.
<code>-nobuild</code>	Parse and validate only
<code>files</code>	Process the files and directories in this list of files. If directories are listed, the command processes all XML files in the listed directories. The names of all files must include an XML suffix. All files must conform to the <code>msgcat.dtd</code> syntax. <code>i18ngen</code> prints the fully-qualified list of names (Java source) to the <code>stdout</code> log for those files actually generated.

I10ngen Utility

The `i10ngen` utility processes the locale-specific catalogs in directories that are subordinate to the top-level catalogs.

Syntax

```
java -classpath <i10n_Classpath> weblogic.i18ntools.i10ngen
[options] filelist
```

where `<i10n_Classpath>` should include `<WebLogic Home>/lib/weblogic.jar`

4 Using the BEA WebLogic Server Internationalization Utilities

Note: Utilities can be run from any directory, but if files are listed on the command line, then their path is relative to the current directory.

Options

Option	Definition
<code>-d targetdirectory</code>	Directory in which to place properties. Default is the current directory.
<code>-language code</code>	Language code. Default is <code>all</code> .
<code>-country code</code>	Country code. Default is <code>all</code> .
<code>-variant code</code>	Variant code. Default is <code>all</code> .
<code>-filelist</code>	Message catalog directories and files to process, relative to the current directory. Identifies top-level, not local-specific, directory.

CatInfo Utility

This utility generates a listing of installed log messages. By default, `CatInfo` lists in order the ID and message body for all currently installed log messages.

Syntax

```
java weblogic.i18ntools.CatInfo [options]
```

Note: Utilities can be run from any directory, but if files are listed on the command line, then their path is relative to the current directory.

Options

Note: All options may be abbreviated to a single character.

Option	Definition
<code>-id nnnnnn</code>	where <i>nnnnnn</i> represents the message ID. The <code>-id</code> option is used to specify a particular message.
<code>-subsystem identifier</code>	The subsystem identifier. The <code>-subsystem</code> option prints only those messages that match the specified subsystem.
<code>-detail</code>	Requests a detailed listing. The <code>-detail</code> option also requests version, severity, subsystem, message detail, cause, and action information.
<code>-help</code>	Provides usage information.

To export the detailed list of messages to a file, use the following command:

```
java weblogic.i18ntools.CatInfo -detail > Errors.txt
```


A Localizer Class Reference for BEA WebLogic Server

The following sections provide reference information for `Localizer` classes:

- [About Localizer Classes](#)
- [Localizer Methods](#)
- [Localizer Lookup Class](#)

Note: This information on `Localizer` class methods is provided as reference for advanced users. Normally, users will not need to use these interfaces directly.

About Localizer Classes

`Localizers` are classes that are used by applications and server code to localize text for output. The `i18ngen` utility creates `Localizer` classes based on the content of the message catalog.

One `Localizer` class is generated for each catalog file. The name of the class is the catalog name (without the `.xml` extension, which is stripped by the utility), followed by `LogLocalizer`. A `Localizer` class for the catalog `ejb.xml` is `ejbLogLocalizer`.

Localizer Methods

Localizers are extensions to the `java.util.ListResourceBundle` class. Four additional methods are provided to simplify the access of the localization data in the `Localizer`. These methods are described in [Table A-1](#).

Table A-1 Methods for Localization Data Access

Method	Description
<code>public Object getObject(String key, String id)</code>	Returns localization text for the <code>key</code> element for message <code>id</code> .
<code>public Object getObject(String key, int id)</code>	Returns localization text for the <code>key</code> element for message <code>id</code> .
<code>public String getString(String key, String id)</code>	Returns localization text for the <code>key</code> element for message <code>id</code> .
<code>public String getString(String key, int id)</code>	Returns localization text for the <code>key</code> element for message <code>id</code> .

Each of the methods for accessing localization data has a `key` argument. The following list shows the recognized values for the `key` argument:

- `Localizer.SEVERITY`
- `Localizer.MESSAGE_ID`
- `Localizer.MESSAGE_BODY`
- `Localizer.MESSAGE_DETAIL`
- `Localizer.CAUSE`
- `Localizer.ACTION`

With the exception of the `Localizer.SEVERITY` key, the localization data returned by Localizers are String objects that return an integer object.

The following list shows the severity values that are returned:

- `weblogic.logging.severities.EMERGENCY`
- `weblogic.logging.severities.ALERT`
- `weblogic.logging.severities.CRITICAL`
- `weblogic.logging.severities.ERROR`
- `weblogic.logging.severities.WARNING`
- `weblogic.logging.severities.NOTICE`
- `weblogic.logging.severities.INFO`
- `weblogic.logging.severities.DEBUG`

The specific strings returned are defined in the message catalogs.

The key argument to the `get*()` methods identify which element of a definition to return. Acceptable values are defined in the `Localizer` class definition. The returned text can be further expanded through `java.text.MessageFormat.format()`. The message body, detail, cause, and action elements are all localizable. The other elements, message ID, severity, and subsystem are not localizable and do not require further processing by `MessageFormat`.

Localizer Lookup Class

To obtain the correct `Localizer` for a message, use the `L10nLookup` class. `L10nLookup` is a `Property` class extension that is loaded at system startup from the property file:

```
\weblogic\i18n\i18n.properties
```

This property file is created by `i18ngen`. Properties in the lookup file have the following format:

```
nnnnnn=subsystem:Localizer class
```

The arguments on this line are defined as follows:

- `nnnnnn` is the message ID
- `subsystem` is the related subsystem

- *Localizer class* is the name of the generated `Localizer` class

For example, message 001234 is identified as an EJB subsystem message ID from the `weblogic.i18n.ejbLogLocalizer` class by the following property in the lookup file:

```
001234=EJB:weblogic.i18n.ejbLogLocalizer
```

B Logger Class Reference for BEA WebLogic Server

The following sections provide reference information for `Logger` classes:

- [About Logger Classes](#)
- [Example of a Generated Logger Class](#)

Note: This information on `Logger` classes is provided as reference of methods for normal usage. Normally, users will not need to use these interfaces directly.

About Logger Classes

The classes generated by `i18ngen` are known as `Loggers`. `Logger` classes provide the interface to WebLogic Server error logging. For catalog `xyz.xml`, a `Logger` class `xyzLogger` is generated.

The `Logger` class provides methods to log all messages defined in a catalog to the WebLogic Server log. The methods included are the same as those defined in the associated catalog. If the catalog specifies the `loggables` attribute to be true, then `Loggable` methods are also generated for each message.

Example of a Generated Logger Class

[Listing B-1](#) show an example of a catalog with a single message definition:

Listing B-1 Example of Message Catalog

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE message_catalog PUBLIC "weblogic-message-catalog-dtd"
"http://www.bea.com/servers/wls600/msgcat.dtd">
<message_catalog
  i18n_package="examples.i18n.logging"
  l10n_package="examples.i18n.logging"
  subsystem="I18N"
  version="1.0"
  baseid="600000"
  endid="610000"
  loggables="true"
>
  <logmessage
    messageid="600000"
    method="logEntry()"
    severity="info"
  >
    <messagebody>Starting I18nLog example...</messagebody>
    <messagedetail></messagedetail>
    <cause></cause>
    <action></action>
  </logmessage>
  <logmessage
    messageid="600001"
    method="testArgs(String name,int cnt)"
    severity="debug"
  >
    <messagebody>Class {0} started with {1,number}
arguments.</messagebody>
    <messagedetail></messagedetail>
    <cause></cause>
    <action></action>
  </logmessage>
  <logmessage
    messageid="600002"
    method="logTrace(Throwable t)"
    severity="error"
```

```
        stacktrace="true"
    >
    <messagebody>This message is followed by a trace</messagebody>
    <messagedetail></messagedetail>
    <cause></cause>
    <action></action>
</logmessage>
<logmessage
  messageid="600003"
  method="logNoTrace(Throwable t)"
  severity="warning"
  stacktrace="false"
  >
    <messagebody>This message is not followed by a trace, but we
can insert its text : {0}</messagebody>
    <messagedetail></messagedetail>
    <cause></cause>
    <action></action>
</logmessage>
<logmessage
  messageid="600004"
  method="getId()"
  severity="info"
  >
    <messagebody>This message's id will be in the next
message</messagebody>
    <messagedetail>A message can contain additional detailed
information.</messagedetail>
    <cause>This message is displayed on purpose</cause>
    <action>Nothing to do, the example is working</action>
</logmessage>
<logmessage
  messageid="600005"
  method="showId(String id)"
  severity="info"
  >
    <messagebody>The previous message logged had message id
{0}</messagebody>
    <messagedetail></messagedetail>
    <cause></cause>
    <action></action>
</logmessage>
</message_catalog>
```

[Listing B-2](#) shows the corresponding java source generated by i18ngen:

Listing B-2 Example of Generated Logger Class

```
package examples.i18n.logging;

import weblogic.logging.MessageLogger;
import weblogic.logging.Loggable;
import java.util.MissingResourceException;

/**
 * Copyright (c) 2001 by BEA Systems, Inc. All Rights Reserved.
 * @exclude
 */
public class I18nLogLogger
{
    /**
     * Starting I18nLog example...
     * @exclude
     *
     * messageid: 600000
     * severity: info
     */
    public static String logEntry() {
        Object [] args = { };
        MessageLogger.log(
"600000",
args,
"examples.i18n.logging.I18nLogLogLocalizer");
        return "600000";
    }
    public static Loggable logEntryLoggable() throws MissingResourceException {
        Object[] args = { };
        return new Loggable("600000", args);
    }
    /**
     * Class {0} started with {1,number} arguments.
     * @exclude
     *
     * messageid: 600001
     * severity: debug
     */
    public static String testArgs(String arg0, int arg1) {
        Object [] args = { arg0, new Integer(arg1) };
        MessageLogger.log(
"600001",
args,
"examples.i18n.logging.I18nLogLogLocalizer");
        return "600001";
    }
}
```

```
public static Loggable testArgsLoggable(String arg0, int arg1) throws
MissingResourceException {
    Object[] args = { arg0, new Integer(arg1) };
    return new Loggable("600001", args);
}
/**
 * This message is followed by a trace
 * @exclude
 *
 * messageid: 600002
 * severity: error
 */
public static String logTrace(Throwable arg0) {
    Object [] args = { arg0 };
    MessageLogger.log(
"600002",
args,
"examples.i18n.logging.I18nLogLogLocalizer");
    return "600002";
}
public static Loggable logTraceLoggable(Throwable arg0) throws
MissingResourceException {
    Object[] args = { arg0 };
    return new Loggable("600002", args);
}
/**
 * This message is not followed by a trace, but we can insert its text : {0}
 * @exclude
 *
 * messageid: 600003
 * severity: warning
 */
public static String logNoTrace(Throwable arg0) {
    Object [] args = { arg0 };
    MessageLogger.log(
"600003",
args,
"examples.i18n.logging.I18nLogLogLocalizer");
    return "600003";
}
public static Loggable logNoTraceLoggable(Throwable arg0) throws
MissingResourceException {
    Object[] args = { arg0 };
    return new Loggable("600003", args);
}
/**
 * This message's id will be in the next message
 * @exclude
 *

```

B *Logger Class Reference for BEA WebLogic Server*

```
* messageId: 600004
* severity: info
*/
public static String getId() {
    Object [] args = { };
    MessageLogger.log(
"600004",
args,
"examples.i18n.logging.I18nLogLogLocalizer");
    return "600004";
}
public static Loggable getIdLoggable() throws MissingResourceException {
    Object[] args = { };
    return new Loggable("600004", args);
}
/**
 * The previous message logged had message id {0}
 * @exclude
 *
 * messageId: 600005
 * severity: info
 */
public static String showId(String arg0) {
    Object [] args = { arg0 };
    MessageLogger.log(
"600005",
args,
"examples.i18n.logging.I18nLogLogLocalizer");
    return "600005";
}
public static Loggable showIdLoggable(String arg0) throws
MissingResourceException {
    Object[] args = { arg0 };
    return new Loggable("600005", args);
}
}
```

[Listing B-3](#) shows an example application that uses the `i18nLog`:

Listing B-3 **Example of Application Using `i18nLog`**

```
package examples.i18n.logging;
```

Example of a Generated Logger Class

```
import java.util.Locale;

import weblogic.i18n.Localizer;
import weblogic.i18ntools.I18nLookup;
import weblogic.logging.Loggable;

/**
 * @author Copyright (c) 2000 by BEA Systems, Inc. All Rights Reserved.
 */

/**
 * This example shows how to use the internationalized (I18n) logging interfaces.
 * <p>
 * usage: java examples.i18n.logging.I18nLog
 * <p>
 * Build procedure: run bld.sh (UNIX) or bld.cmd (NT). These scripts
 * process the I18nLog.xml catalog, producing the logging class,
 * <tt>examples.i18n.logging.I18nLogLogger</tt>. This class contains static
 * methods
 * for logging messages to the WLS error log. The methods and arguments are
 * defined in the I18nLog.xml catalog. This example also uses a simple
 * message catalog, I18nSimple.xml.
 */

public class I18nLog {

    public I18nLog() {}

    public static void main(String[] argv) {
        /**
         * This call just logs an info message. There are no arguments defined
         * for this method.
         *
         * This also shows how to use the Loggable form of the method.
         */

        Loggable ll = I18nLogLogger.logEntryLoggable();
        ll.log();
        System.out.println(ll.getMessage());

        /**
         * Here's an example of a message including a variety
         * of arguments.
         */
        I18nLogLogger.testArgs(I18nLog.class.getName(), argv.length);
        /**
         * If a Throwable is passed then it will result in a stack trace
         * being logged along with the method by default.
         */
    }
}
```

B *Logger Class Reference for BEA WebLogic Server*

```
*/
Throwable t = new Throwable("Test with stack trace");
I18nLogLogger.logTrace(t);
/**
 * Messages can optionally be defined to not log a stack trace.
 */
I18nLogLogger.logNoTrace(t);
/**
 * The logger methods return the messageId for applications
 * that want to do more than just log these messages.
 */
String messageId = I18nLogLogger.getId();
I18nLogLogger.showId(messageId);
/**
 * the message id can be used to obtain the different attributes
 * of a message. The L10nLookup object provides access to the catalogs
 * via Localizer classes. Localizers provide the access to individual
 * messages. Each log message catalog has two Localizers: one for
 * general message information and one for the detailed attributes.
 *
 * The basic Localizer provides access to catalog information:
 *   Version
 *   L10n Package - package for catalog data
 *   I18n Package - package for Logger methods
 *   Subsystem - catalog subsystem
 * For each message it also provides:
 *   Severity: debug (128), info (64), warning (32), error (8)
 *   Message Body - the message text
 *   Stack option - whether to log a stack trace
 *
 * First get to the L10nLookup properties, then use them to get the
 * Localizer's for the message.
 */
L10nLookup l10n = L10nLookup.getL10n();
/**
 * This returns the basic Localizer (arg 3 = false)
 */
Localizer lcl = l10n.getLocalizer(messageId, Locale.getDefault(), false);
/**
 * This returns the detailed Localizer (arg 3 = true)
 */
Localizer lclDetail = l10n.getLocalizer(messageId, Locale.getDefault(), true);
/**
 * Use this applications simple message catalog to display the
 * log message catalog information
 */
I18nSimpleTextFormatter fmt = new I18nSimpleTextFormatter();
System.out.println(fmt.version(messageId, lcl.getVersion()));
System.out.println(fmt.l10nPackage(messageId, lcl.getL10nPackage()));
```

```
System.out.println(fmt.i18nPackage(messageId, lcl.getI18nPackage()));
System.out.println(fmt.subsystem(messageId, lcl.getSubSystem()));
System.out.println(fmt.severity(messageId, lcl.getSeverity(messageId)));
System.out.println(fmt.body(messageId, lcl.getBody(messageId)));
System.out.println(fmt.stack(messageId, lcl.getStackTrace(messageId)));
/**
 * Now for the detailed information.
 */
System.out.println(fmt.detail(messageId, lclDetail.getDetail(messageId)));
System.out.println(fmt.cause(messageId, lclDetail.getCause(messageId)));
System.out.println(fmt.action(messageId, lclDetail.getAction(messageId)));
}
}
```

[Listing B-2](#) shows the corresponding java source generated by `i18ngen`:

Listing B-4 Example of Generated Logger Class

C Loggable Object Reference for BEA WebLogic Server

The following sections provide reference information for `Loggable` objects:

- [About Loggable Objects](#)
- [How Loggable Objects Are Used](#)

About Loggable Objects

By default, all log message catalogs create `Logger` classes with methods that are used to log the messages to the WebLogic Server log. The `Logger` classes can optionally include methods that return a `Loggable` object instead of logging the message. `Loggable` objects are useful when you want to generate the log message but actually log it at a later time. They are also useful if you want to use the message text for other purposes, such as throwing an exception.

How Loggable Objects Are Used

To create a `Logger` that provides methods to return `Loggable` objects, you need to set the `loggables` attribute in the message catalog.

For example, consider the `test.xml` catalog shown in [Listing C-1](#).

Listing C-1 `test.xml` Message Catalog

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE message_catalog PUBLIC "weblogic-message-catalog-dtd"
"http://www.bea.com/servers/wls600/msgcat.dtd">
<message_catalog
  subsystem="Examples"
  version="1.0"
  baseid="500000"
  endid="500001"
  loggables="true"
>
  <logmessage
    messageid="500000"
    severity="error"
    method="logIOError(Throwable t)"
  >
    <messagebody>
      IO failure detected.
    </messagebody>
    <messagedetail>
    </messagedetail>
    <cause>
    </cause>
    <action>
    </action>
  </logmessage>
</message_catalog>
```

When you run this catalog through the `il8ngen` utility, a `Logger` class is created for this catalog with the following two methods:

- `logIOError (Throwable)`—logs the message

- `logIOExceptionLoggable (Throwable)`—returns a `Loggable` object

The `Loggable` class can be used as shown in [Listing C-2](#).

Listing C-2 Example of Use of Loggable Class

```
package test;
import weblogic.logging.Loggable;
import weblogic.i18n.testLogger;

...
try {
    // some IO
} catch (IOException ioe) {
    Loggable l = testLogger.logIOExceptionLoggable(ioe);
    l.log(); // log the error
    throw new Exception(l.getMessage()); // throw new exception with
        same text as logged
}
```

D TextFormatter Class Reference for BEA WebLogic Server

The following sections provide reference information for `TextFormatter` classes:

- [About TextFormatter Classes](#)
- [Example of an Application Using a TextFormatter Class](#)

Note: This information on `TextFormatter` classes is provided as reference of methods for normal usage. Normally, users will not need to use these interfaces directly.

About TextFormatter Classes

`TextFormatter` classes are generated by `i18ngen` from simple message catalogs. These classes provide methods for generating localized versions of message text at runtime. The following section shows an example of an application, its simple message catalog, and the `TextFormatter` class generated for the catalog.

Example of an Application Using a TextFormatter Class

[Listing 4-3](#) shows an example of a simple message catalog for the HelloWorld application.

Listing 4-3 Example of a Simple Message Catalog

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

<!DOCTYPE message_catalog PUBLIC "weblogic-message-catalog-dtd"
"http://www.bea.com/servers/wls600/msgcat.dtd">
<message_catalog

    i18n_package="examples.i18n.simple"

    subsystem="I18N"

    version="1.0"

>

<message
    messageid="HELLO_WORLD"
    datelastchanged="967575717875"
    method="helloWorld() "
    >
    <messagebody>
        Hello World!
    </messagebody>
</message>

<!-- -->
<message
    messageid="HELLO_AGAIN"
    datelastchanged="967575717804"
    method="helloAgain() "
    >
    <messagebody>
        Hello again
    </messagebody>
```

```
</message>

<!-- -->
<message
  messageid="NTH_HELLO"
  datelastchanged="967575770971"
  method="nthHello(int count)"
  >
  <messagebody>
    This is hello number {0,number}.
  </messagebody>
</message>

<!-- -->
<message
  messageid="VERSION"
  datelastchanged="967578656214"
  method="version(String version)"
  >
  <messagebody>
    Catalog version: {0}
  </messagebody>
</message>

<!-- -->
<message
  messageid="I18N_PACKAGE"
  datelastchanged="967578691394"
  method="i18nPackage(String pkg)"
  >
  <messagebody>
    I18n Package: {0}
  </messagebody>
</message>

<!-- -->
<message
  messageid="L10N_PACKAGE"
  datelastchanged="967578720156"
  method="l10nPackage(String pkg)"
  >
  <messagebody>
    L10n Package: {0}
  </messagebody>
</message>

<!-- -->
<message
  messageid="SUBSYSTEM"
```

```
    datelastchanged="967578755587"
    method="subSystem(String sub)"
  >
  <messagebody>
    Catalog subsystem: {0}
  </messagebody>
</message>
</message_catalog>
```

[Listing 4-4](#) shows an example of an application using the HelloWorld catalog.

Listing 4-4 Example of an Application Using the HelloWorld Catalog

```
package examples.i18n.simple;

import java.util.Locale;
import java.text.MessageFormat;

import weblogic.i18n.Localizer;
import weblogic.i18ntools.L10nLookup;

/**
 * @author Copyright (c) 2000 by BEA Systems, Inc. All Rights
 * Reserved.
 */

/**
 * This example shows various ways of internationalizing an
 * application
 * using simple message catalogs.
 * <p>
 * Usage: java examples.i18n.simple.HelloWorld [lang [country]]
 * <p>
 * lang is a 2 character ISO language code. e.g. "en"
 * country is a 2 character ISO country code. e.g. "US"
 * <p>
 * Usage of any of the languages supported by this example presumes
 * the existence of the appropriate OS localization software and
 * character
 * encodings.
 * <p>
 * The example comes with catalogs for English (the default) and
 * French.
 * The catalog source is in the following files, and were built
```

Example of an Application Using a TextFormatter Class

```
* using the catalog editing utility,
weblogic.il8ntools.gui.MessageEditor.
* <p>
* <pre>
* English(base language)      ../msgcat/Helloworld.xml
* French                      ../msgcat/fr/FR/HelloWorld.xml
* </pre>
* <p>
* To build this example run the bld.sh(UNIX) or bld.cmd (NT)
scripts from
* the examples/il8n/simple directory. CLIENT_CLASSES must be set
up and
* needs to be in the classpath when running the example.
*/
```

```
public final class HelloWorld {

    public static void main(String[] argv) {
        /*
         * The easiest method for displaying localized text is to
         * instantiate the generated formatter class for the HelloWorld
         catalog.
         * This class contains convenience methods that return localized
         text for
         * each message defined in the catalog. The class name is
         * the catalog name followed by "TextFormatter".
         *
         * Normally, one would use the default constructor to obtain
         * formatting in the current locale. In this example we'll use
         a locale
         * based on arguments to construct the TextFormatter.
         */
        Locale lcl;
        if (argv.length == 0) { // default is default locale for JVM
            lcl = Locale.getDefault();
        }
        else {
            String lang = null;
            String country = null;
            //get the language code
            lang = argv[0];
            if (argv.length >= 2) { // get the country code
                country = argv[1];
            }
            lcl = new Locale(lang, country);
        }
        /*
         * get formatter in appropriate locale
        */
    }
}
```

D *TextFormatter Class Reference for BEA WebLogic Server*

```
    */
    HelloWorldTextFormatter fmt = new HelloWorldTextFormatter(lcl);
    fmt.setExtendedFormat(true);
    /*
     * print the text in the current locale
     */
    System.out.println(fmt.helloWorld());
    /*
     * Alternatively, text can be accessed and formatted manually.
    In this
     * case you must obtain the Localizer class for the catalog.
    The Localizer
     * class is formed from the l10n_package attribute in the
    catalog, the
     * catalog name, and the string "TextLocalizer".
     */
    Localizer l10n = L10nLookup.getLocalizer
        (lcl, "examples.i18n.simple.HelloWorldTextLocalizer");
    System.out.println(l10n.get("HELLO_AGAIN"));
    /*
     * If the message accepts arguments, then they can just be passed
    to the
     * method defined for the message.
     */
    System.out.println(fmt.nthHello(3));
    /*
     * If using the manual method then you must manually apply the
    argument to
     * the text using the MessageFormat class.
     */
    String text = l10n.get("NTH_HELLO");
    Object[] args = {new Integer(4)};
    System.out.println(MessageFormat.format(text, args));
    /*
     * The Localizer class also provides methods for accessing
    catalog information.
     */
    System.out.println(fmt.version(l10n.getVersion()));
    System.out.println(fmt.l10nPackage(l10n.getL10nPackage()));
    System.out.println(fmt.i18nPackage(l10n.getI18nPackage()));
    System.out.println(fmt.subSystem(l10n.getSubSystem()));
}
}
```

Listing 4-5 shows an example of the generated `TextFormatter` for the `HelloWorld` catalog.

Listing 4-5 Example of Generated TextFormatter Class for the HelloWorld Catalog

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

<!DOCTYPE message_catalog PUBLIC "weblogic-message-catalog-dtd"
"http://www.bea.com/servers/wls600/msgcat.dtd">
<message_catalog

    l10n_package="examples.i18n.simple"

    subsystem="I18N"

    version="1.0"

    >

    <message
        messageid="HELLO_WORLD"
        datelastchanged="967575717875"
        method="helloWorld()"
        >
        <messagebody>
            Hello World!
        </messagebody>
    </message>

    <!-- -->
    <message
        messageid="HELLO_AGAIN"
        datelastchanged="967575717804"
        method="helloAgain()"
        >
        <messagebody>
            Hello again
        </messagebody>
    </message>

    <!-- -->
    <message
        messageid="NTH_HELLO"
        datelastchanged="967575770971"
        method="nthHello(int count)"
        >
        <messagebody>
            This is hello number {0,number}.
        </messagebody>
    </message>
```

```
<!-- -->
<message
  messageid="VERSION"
  datelastchanged="967578656214"
  method="version(String version)"
  >
  <messagebody>
    Catalog version: {0}
  </messagebody>
</message>

<!-- -->
<message
  messageid="I18N_PACKAGE"
  datelastchanged="967578691394"
  method="i18nPackage(String pkg)"
  >
  <messagebody>
    I18n Package: {0}
  </messagebody>
</message>

<!-- -->
<message
  messageid="L10N_PACKAGE"
  datelastchanged="967578720156"
  method="l10nPackage(String pkg)"
  >
  <messagebody>
    L10n Package: {0}
  </messagebody>
</message>

<!-- -->
<message
  messageid="SUBSYSTEM"
  datelastchanged="967578755587"
  method="subSystem(String sub)"
  >
  <messagebody>
    Catalog subsystem: {0}
  </messagebody>
</message>
</message_catalog>
```

Index

A

argument
 key A-2
 message 2-4

C

catalog
 browsing for 3-5
 context 3-2
 creating 3-7
 entering a new log message 3-9
 entering a simple text message 3-11
 locale-specific 1-5, 2-3, 2-5
 message 1-4
 naming 2-3
 top-level 2-2, 2-5
CatInfo 4-6
character code 3-2
class
 Localizer A-1
 Loggable C-3
 Logger 4-2, B-1, C-2
 TextFormatter 4-2, D-1
context catalog 3-2
customer support contact information ix

D

documentation, where to find it viii
DTDs 2-2

E

elements
 locale-specific catalog 2-14
 log message catalog 2-5
 simple text message catalog 2-10

I

I18n package 3-8
i18ngen 4-4
Internationalization
 definition of 1-1
Internationalization Interfaces
 Java 1-4

J

Java Development Kit (JDK) 1-4
Java internationalization interfaces 1-4

K

key argument A-2

L

L10n package 3-8
l10n_msgcat.dtd 2-2
l10ngen 4-5
L10nLookup A-3
locale-specific catalog 2-14
Localization

- definition of 1-2
- log messages 1-2
- simple text 1-2
- Localizers 4-2, A-1
- log message catalog
 - elements 2-5
 - example 2-9
 - syntax 2-15
- Loggable object C-1
- Logger B-1

M

- message
 - arguments 2-4
 - create an internationalized 1-3
 - editing 3-16
 - finding 3-12
 - finding a log message 3-12
 - selecting in Message Viewer 3-15
 - viewing all in catalog 3-14
 - viewing all in several catalogs 3-15
- message catalog
 - formats 2-5
 - hierarchy 2-2
 - naming 2-3
- message editor
 - about 3-1
 - starting 3-3
- Message IDs 1-3
- Message Viewer 3-14
- message, finding a simple text message 3-13
- method
 - Localizer A-2
 - Logger 1-5
 - TextFormatter 1-5
- msgcat directory 2-2
- msgcat.dtd 2-2

N

- naming conventions
 - classes 2-3
 - message catalogs 2-3

O

- object
 - Loggable C-1

P

- printing product documentation viii
- property file A-3

R

- related information ix

S

- severity values A-3
- stacktrace 3-11
- Standards
 - Internationalization and Localization 1-1
- Subsystem 3-8
- support
 - technical ix

U

- utilities
 - internationalization and localization 4-1