BEA WebLogic Server™

Internationalization Guide

Version 8.1
Revised: March 24, 2003

Copyright

Restricted Rights Legend

Trademarks or Service Marks

Contents

About This Document

1. Overview of Internationalization and Localization for WebLogic Server

2. Using Message Catalogs with BEA WebLogic Server

3. Using the BEA WebLogic Server Message Editor

C. TextFormatter Class Reference for BEA WebLogic Server

D. Logger Class Reference for BEA WebLogic Server

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 and Localization 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 D, "Logger Class Reference for BEA WebLogic Server," describes `Logger` classes and provides an example of a message catalog and its corresponding `Logger` class.
- Appendix B, "Loggable Object Reference for BEA WebLogic Server," describes `loggable objects` and how they are used.

- Appendix C, "TextFormatter Class Reference for BEA WebLogic Server," provides and 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.
italics	Emphasis and book titles.

Convention	Usage

`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.
	Examples:
	`import java.util.Enumeration;`
	`chmod u+w *`
	`config/examples/applications`
	`.java`
	`config.xml`
	`float`

`monospace italic text`	Variables in code. *Example*: `String CustomerName;`
UPPERCASE TEXT	Device names, environment variables, and logical operators. *Example*s: LPT1 BEA_HOME OR
`{ }`	A set of choices in a syntax line.
`[ ]`	Optional items in a syntax line. *Example*: `java utils.MulticastTest -n name -a address [-p portnumber] [-t timeout] [-s send]`
`|`	Separates mutually exclusive choices in a syntax line. *Example*: `java weblogic.deploy [list|deploy|undeploy|update] password {application} {source}`
`...`	Indicates one of the following in a command line: • 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

CHAPTER 1

Overview of Internationalization and Localization for WebLogic Server

The following sections provide an overview of localization and internationalization:

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.

Understanding Localization and Internationalization 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. For more information, see "Elements of a Log Message Catalog" on page 2-6.
- **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. For more information, see "Elements of a Simple Text Catalog" on page 2-14.

Understanding Message Catalogs

All internationalized text is defined in message catalogs, each of which defines a collection of log messages or 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 WebLogic Server Administration Console is simple text. As with log messages, simple text can be merged with dynamic data.

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. For more information, see "Elements of a Log Message Catalog" on page 2-6.
- **Simple text catalogs**—Collections of simple text messages. For more information, see "Elements of a Simple Text Catalog" on page 2-14.

- **Locale message catalogs**—Collections of locale-specific messages corresponding to a top-level log message or simple text catalog. For more information, see "Elements of a Locale-Specific Catalog" on page 2-18.

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.

To view the WebLogic Server message catalogs, refer to the "Index of Messages by Message Range."

Understanding Java Interfaces for Internationalization

WebLogic Server uses the Java internationalization interfaces to provide 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
`java.util.Locale`	Represents a specific geographical, political, or cultural region.
`java.util.ResourceBundle`	Provides containers for locale-specific objects.
`java.text.MessageFormat`	Produces concatenated messages in a language-neutral way.

Main Steps for Creating an Internationalized Message

The following steps summarize how you create an internationalized message to use with WebLogic Server. Later sections of this guide describe these steps in more detail.

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.

2. Run `weblogic.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 class is defined according to information specified in the message catalog entry. The classes include `Logger` or `TextFormatter` methods, depending on the type of catalog. For details, see "weblogic.i18ngen Utility" on page 4-3.

3. Create locale-specific catalogs as required for the message catalog you created in Step 1. For more information, see "Example Locale-Specific Catalog" on page 2-17 in Chapter 2, "Using Message Catalogs with BEA WebLogic Server."

4. Run `weblogic.l10ngen` to process the locale-specific catalogs. For details, see "weblogic.l10ngen Utility" on page 4-6.

5. Configure the application to use the `Logger` or `TextFormatter` classes 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` classes used.

For more detailed information, including an overview of the logging subsystem and a description of log message parts, see "Writing Messages to the WebLogic Server Log" in *Using WebLogic Logging Services*.

CHAPTER 2

Using Message Catalogs with BEA WebLogic Server

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

Overview of Message Catalogs

A message catalog is a single XML file that contains a collection of text messages, with each message indexed by a unique idenifier. You compile these XML files into classes during the `weblogic.i18ngen` utility build process. (Refer to "weblogic.i18ngen Utility" on page 4-3 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, which contains the English version of the messages. Then there are corresponding locale-specific catalogs, one for each additional supported locale.

The top-level catalog (English version) 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 `weblogic/msgcat` directory of `WL_HOME/server/lib/weblogic.jar`.

You can also download the DTDs at:
`http://www.bea.com/servers/wls810/dtd/msgcat.dtd`.

Two DTDs are included in the WebLogic Server distribution:

- `msgcat.dtd`—Describes the syntax of top-level, default catalogs.
- `l10n_msgcat.dtd`—Describes the syntax of locale-specific catalogs.

The `weblogic/msgcat` directory of `WL_HOME/server/lib/weblogic.jar` contains templates that you can use to create top-level and locale-specific message catalogs.

You can create a single log message catalog for all logging requirements, or create smaller catalogs based on a subsystem or Java package. BEA recommends using multiple subsystems because you can focus on specific portions of the log during viewing.

For simple text catalogs, BEA recommends creating a single catalog for each utility being internationalized. You create 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 must be defined in the default, top-level catalog. The WebLogic Server distribution includes a collection of sample catalogs in the *SAMPLES_HOME*`/server/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. However, English is not required for any catalogs, except for those in the *SAMPLES_HOME*`/server/examples/i18n/msgcat/` directory.

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.

Guidelines for Naming 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 the names of existing classes in the target package for which you are creating the message catalog.
- 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.

  Note: This only applies to log message catalogs. Simple text catalogs can have any string value.

- 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 (Message Body, Message Detail, Probable Cause), although the message body must 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. In Listing 2-1, {0} represents the file that is cannot be opened, while {1} represents the file that will be opened in its place.

Listing 2-1 Example of Message Arguments

```
<messagebody>Unable to open file, {0}. Opening {1}. All arguments must
be in body.</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 is as follows:

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

The message example in Listing 2-1 expects two arguments, {0} and {1}:

- Both are used in the <messagebody>.
- Both are used in the <messagedetail>.
- Neither is used in <cause> or <action>.

Note: 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 (message body, message detail, cause, action), although the message body must include all of the arguments.

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 (by default, this is the English language). Locale-specific catalogs (for example, those translated to Spanish) only provide translations of text defined in the top-level version. Log message catalogs are defined differently from simple text catalogs.

Examples and elements of each type of message catalog are described in the following sections:

Example Log Message Catalog

The following example shows a log message catalog, `MyUtilLog.xml`, containing one log message. This log message illustrates the usage of the `messagebody`, `messagedetail`, `cause` and `action` elements.

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/wls810/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 approriate authorization or learn to
      live with the default settings.
    </action>
  </log_message>
</message_catalog>
```

Elements of a Log Message Catalog

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

- "message_catalog Element" on page 2-7
- "log_message Element" on page 2-9
- "Child Elements of log_message Catalog Element" on page 2-12

message_catalog Element

The `message_catalog` element represents the log message catalog. The following table describes the attributes that you can define for the `message_catalog` element.

Attribute	Default Value	Required/ Optional	Description
`i18n_package`	`weblogic.i18n`	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 `mycat.xml`, a generated logger class would be called *<i18n_package>*`.mycatLogger.class`. Syntax: standard Java package syntax Example: `i18n_package="programs.utils"`
`l10n_package`	`weblogic.i18n`	Optional	A Java package containing generated `LogLocalizer` properties for the catalog. For example, for a catalog called `mycat.xml`, the following property files would be generated: *<l10n_package>*`.mycatLogLocalizer.properties` and `l10n_package>mycatLogLocalizerDetail.properties`. Syntax: standard Java package syntax Example: `i18n_package="programs.utils"`
`subsystem`	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. Syntax: a String Example: `subsystem="MYUTIL"`

Attribute	Default Value	Required/ Optional	Description
`version`	None	Required	Specifies the version of the `msgcat.dtd` being used. Use: Must be `"1.0"`. Syntax: x.y where x and y are numeric. Example: `version="1.0"`
`baseid`	`000000` for WebLogic Server catalogs `500000` for user-defined catalogs	Optional	Specifies the lowest message ID used in this catalog. Syntax: one to six decimal digits. Example: `baseid="600000"`
`endid`	`499999` for WebLogic Server catalogs `999999` for user-defined catalogs	Optional	Specifies the highest message ID used in this catalog. Syntax: one to six decimal digits. Example: `endid="600100"`
loggable	false	Optional	Indicates whether to generate additional methods that return loggable objects. Syntax: `"true"` or `"false"` Example: `loggable="true"`

Attribute	Default Value	Required/ Optional	Description
prefix	Null for user-defined catalogs. "BEA" for WebLogic Server catalogs.	Optional	Specifies a String to be prepended to message IDs when logged. Server messages default to "BEA" as the prefix and may not specify a different prefix. User messages can specify any prefix. A prefixed message ID is presented in a log entry as follows: <[prefix-]id> where `prefix` is this attribute and `id` is the six-digit message ID associated with a specific message. For example, if `prefix` is "XYZ", then message 987654 would be shown in a log entry as <XYZ-987654>. If the prefix is not defined, then the log entry would be <987654>. Syntax: any String (should be limited to five characters) Example: `prefix="BEA"`
description	Null (no description)	Optional	An optional attribute that serves to document the catalog content. Syntax: any String Example: `description="Contains messages logged by the foobar application"`

log_message Element

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

Attribute	Default Value	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. Use: Value must be in the range defined by the `baseid` and `endid` attributes defined in the message_catalog attribute. Syntax: one to six decimal digits. Example: `messageid="600001"`
`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: `Long.toString(new Date().getTime());` Use: The date is supplied by utilites (such as MessageEditor), which run on the catalogs Syntax: `Long.toString(new Date().getTime());`
`severity`	None	Required	Indicates the severity of the log message. Must be one of the following: `debug`, `info`, `warning`, `notice`, `error`, `critical`, `alert`, or `emergency`. User-defined catalogs may only use `debug`, `info`, `warning`, and `error`. Example: `severity="warning"`

Attribute	Default Value	Required/ Optional	Description
`method`	None	Required	Method signature for logging this message. 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 `java.lang`. Classes must also conform to `java.text.MessageFormat` conventions. In general, class arguments should have a useful `toString()` method. Arguments can be any valid name, but should follow the convention of `argn` where `n` is 0 thru 9. There can be no more than 10 arguments. For each `argn` there should be at least one corresponding placeholder in the text elements described in "Child Elements of log_message Catalog Element" on page 2-12. Placeholders are of the form `{n}`, `{n,number}` or `{n,date}`.
methodtype	`logger` (indicating the method generated will log the message)	Optional	Specifies type of method to generate. Methods can be loggers or getters. Logger methods format the message body into the default locale and log the results. Getter methods return the message body prefixed by the subsystem and messageid, as follows: `[susbsystem:msgid]text` Syntax: values are "`logger`" and "`getter`"
`stacktrace`	`true`	Optional	Indicates whether to generate a stack trace for Throwable arguments. Possible values are `true` or `false`. When the value is `true`, a trace is generated. Syntax: `stacktrace="true"`
retired	`false`	Optional	Indicates whether message is retired. A retired message is one that was used in a previous release but is now obsolete and not used in the current version. Retired messages are not represented in any generated classes or resource bundles. Syntax: values are "`true`" and "`false`" Example: `retired="true"`

Child Elements of log_message Catalog Element

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

Element	Parent Element	Required/ Optional	Description
`messagebody`	`log_message`	Required	A short description for this message. The `messagebody` element can contain a 0 to 10 placeholder as {n}, to be replaced by the appropriate argument when the log message is localized. The message body must include placeholders for all arguments listed in the corresponding method attribute, unless the last argument is throwable or a subclass. Be careful when using single quotes, because these are specially parsed by `java.text.MessageFormat`. If it is appropriate to quote a message argument, use double quotes (as in the first example below). If a message has one or more placeholders, in order for a single quote to appear correctly (for example, as an apostrophe), it must be followed by a second single quote. See the example below. Syntax: `String` Example: <messagebody>Could not open file "{0}" created on {1,date}. </messagebody>
`messagedetail`	`log_message`	Optional	A detailed description of the event. This element may contain any argument place holders. Syntax: `String` Example: <messagedetail>The configuration for this application will be defaulted to factory settings.</messagedetail>

Element	Parent Element	Required/ Optional	Description
cause	log_message	Optional	The root cause of the problem. This element can contain any argument place holders. Syntax: a String Example: <cause>The user is not authorized to use custom configurations. The attempt has been logged to the security log.</cause>
action	log_message	Optional	The recommended resolution. This element can contain any argument place holders. Syntax: a String Example: <action>The user needs to gain approriate authorization or learn to live with the default settings.</action>

Example Simple Text Catalog

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

```
<messagebody>
    File
</messagebody>
```

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/wls810/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 Simple Text Catalog

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

message_catalog Element

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

Attribute	Default Value	Required/ Optional	Description
l10n_package	weblogic.i18n	Optional	Java package containing generated TextFormatter classes and TextLocalizer properties for this catalog. The classes are named after the catalog file name. mycat.xml would have the properties file, <l10n_package>.mycatLogLocalizer.properties generated. Syntax: standard Java package syntax Example: i18n_package="programs.utils"
subsystem	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. Syntax: a String Example: subsystem="MYUTIL"
version	None	Required	Specifies the version of the msgcat.dtd being used. The format is *n.n*, for example, version="1.0". Must be at least "1.0". Example: version="1.0"
description	Null	Optional	An optional attribute that documents the catalog content. Syntax: any String Example: description="Contains labels used in the foobar GUI"

message Element

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

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

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

messagebody Element

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.

Example Locale-Specific Catalog

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/wls810/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.

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 Element

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

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

log_message Element

The locale-specific catalog uses the attributes defined for the `log_message` element in the top-level 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 top-level log message catalog so these elements do not need to be defined.

CHAPTER 3

Using the BEA WebLogic Server Message Editor

The following sections describe how to use the Message Editor:

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 or any XML editing tool.

Note: WebLogic Server provides its own message catalogs, which contain messages relating to WebLogic Serversubsystems and functionality. You cannot edit these catalogs. For descriptions of WebLogic Server catalogs, see the *Index of Messages by Message Range*.

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

You use the Message Editor 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
- Retire and unretire messages

Starting the Message Editor

Before you start the Message Editor, install and configure your WebLogic Server system and set your environment variables (`setExamplesEnv.cmd`). Make sure that your classpath is set correctly. For more information on these topics, see the *Installation Guide*.

Sample message catalog files are located in your *SAMPLES_HOME*`/server/examples/i18n/msgcat/` directory.

You can use the sample message catalogs as templates to create your own messages. You simply modify the provided information, such as the package name and class name. Then translate the message text and save the catalog. For more information on this topic, see "Writing Messages to the WebLogic Server Log" in *Using WebLogic Logging Services*.

Note: The location of your `samples/` 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
```

To access basic command line help, type:

java weblogic.MsgEditor -help

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

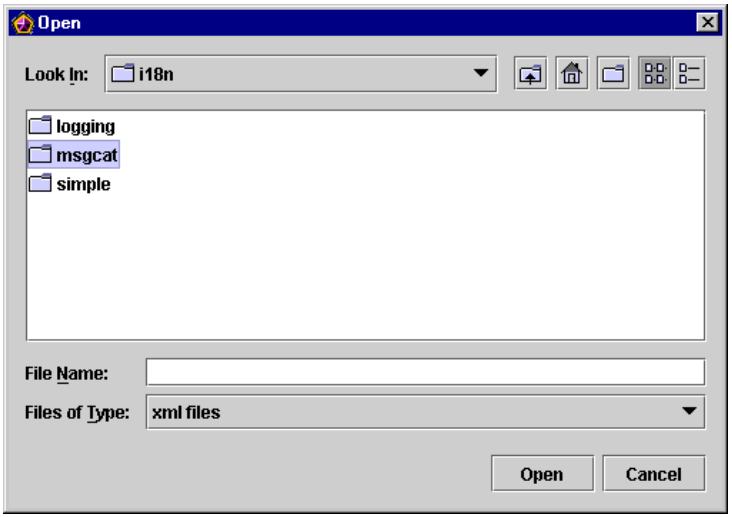
Working with Catalogs

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

Browsing to an Existing Catalog

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 Navigating to a Catalog



The sample catalogs included with your WebLogic Server installation are in the *SAMPLES_HOME*/server/examples/i18n/msgcat/ directory.

Note: This directory path may vary, depending on where you installed WebLogic Server.

You can place your user-defined catalogs in any directory you designate.

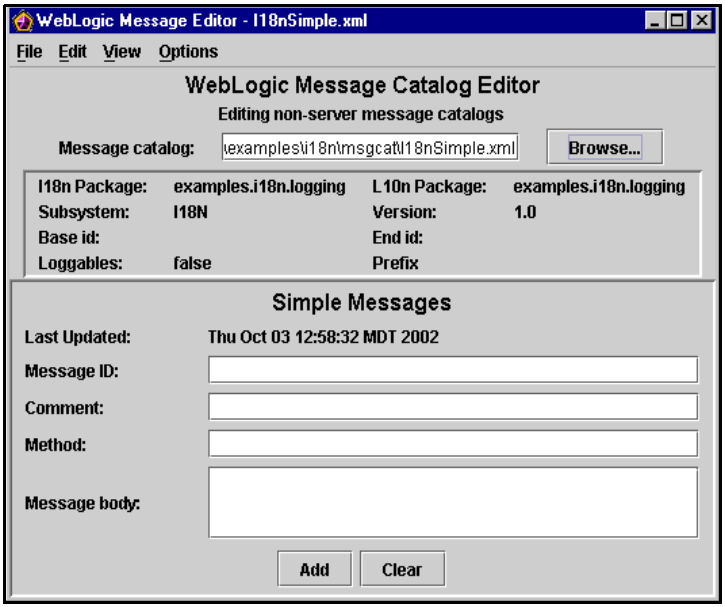
Once you locate the **Packages**, **Subsystem**, **Version**, **Base ID**, and **End ID** (if any) for that catalog are displayed, and that catalog is the context catalog in which all other actions are performed. You are now ready to enter new messages into that catalog, edit existing messages, search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



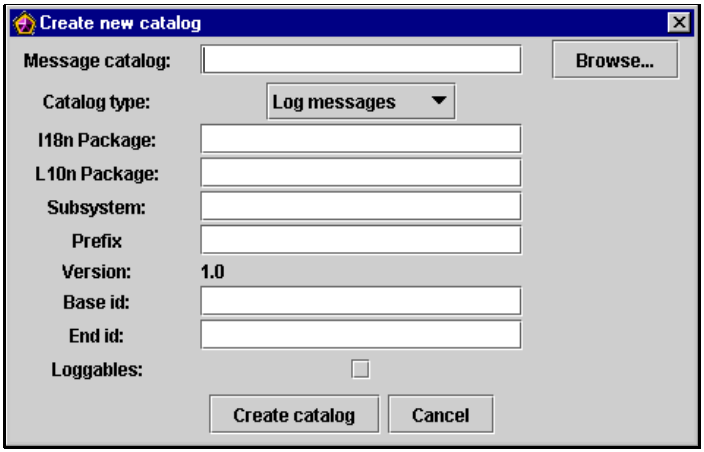
Creating a New Catalog

To create a new catalog, complete the following procedure:

1. From the main menu bar of the WebLogic Message Editor window, choose **File > New Catalog**.

   The "Create new catalog" dialog displays as shown in Figure 3-5.

Figure 3-5 Create New Catalog



2. 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 appropriate catalog directory. (This would be the `msgcat` directory if you are using WebLogic Server example messages.)

3. 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.

4. 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.

5. Enter the name of the package where you want to place the catalog data in the **L10n Package** field. The default is `weblogic.l10n`. If you want to place your catalog data in another package with your application, specify the package name here.

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

7. In the **Prefix** field, enter a prefix to be prepended to the message ID when logged. The default server message prefix is BEA. You can enter any prefix for user messages. (BEA recommends that the prefix be less that 10 characters in length. Also, make sure you use standard java naming conventions.)
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" on page 3-8
- "Entering a New Simple Text Message" on page 3-10

Entering a New Log Message

To enter a new message into a log catalog:

1. From the WebLogic Message Editor main dialog (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. Click **Get next ID** to generate the next unique numerical ID in the context catalog. This ID will appear in the **Message ID** field.
3. Enter any relevant comments about the message in the **Comment** field.
4. Enter the appropriate **Method** for your log message, including parentheses and any arguments. For example: `logErrorSavingTimestamps(Exception ioExcep)`
5. Set the **Method Type** for the log message. Your choices are `logger` and `getter`. The default method type is `logger`, which is used for messages that will be logged. The `getter` choice is for messages that are used for non-logging purposes, such as exceptions.
6. Choose a **Severity** from the list of possible levels.

7. Enter text for the **Message body.** Parameters are denoted by `{n}`. **For example:** **`"Exception occurred while loading _WL_TIMESTAMP FILE."`**
8. Enter text for the **Message detail.** Parameters are denoted by `{n}`. **For example:** **`"Exception occurred while loading _WL_TIMESTAMP FILE. Forcing recompilation: {0}."`**
9. Enter text for the **Probable Cause.** Parameters are denoted by `{n}`. **For example:** **`"There was an error reading this file."`**
10. Enter text for the **Action.** Parameters are denoted by `{n}`. **For example:** **`"No action required."`**
11. 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.
12. Toggle the **Retired message** option on or off by clicking the checkmark box. Use this option to retire (hide) obsolete messages. Retired messages are deleted in the sense that they are not represented in the generated classes. However, the message data does remain in the `.xml` file.
13. Click **Add**.

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

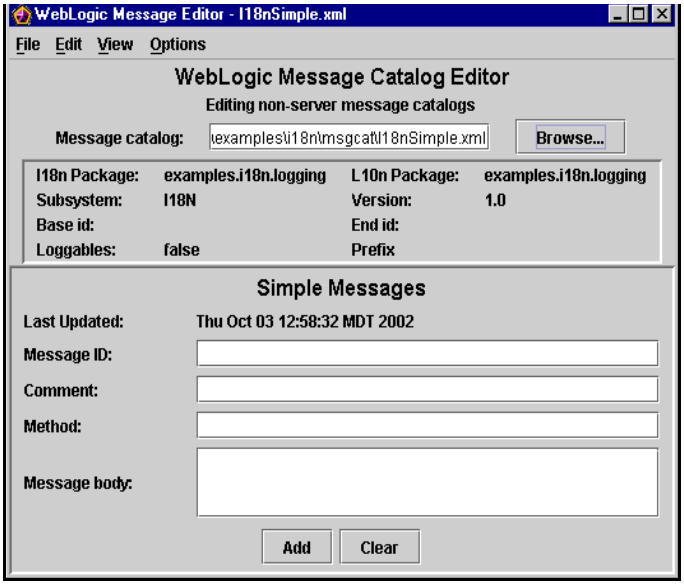
Entering a New Simple Text Message

To enter a simple text message into a simple messages catalog:

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. For example: `startingClusterService()`
5. Enter the **Message body** text. For example: `startingClusterService`
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

To find a log message:

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:

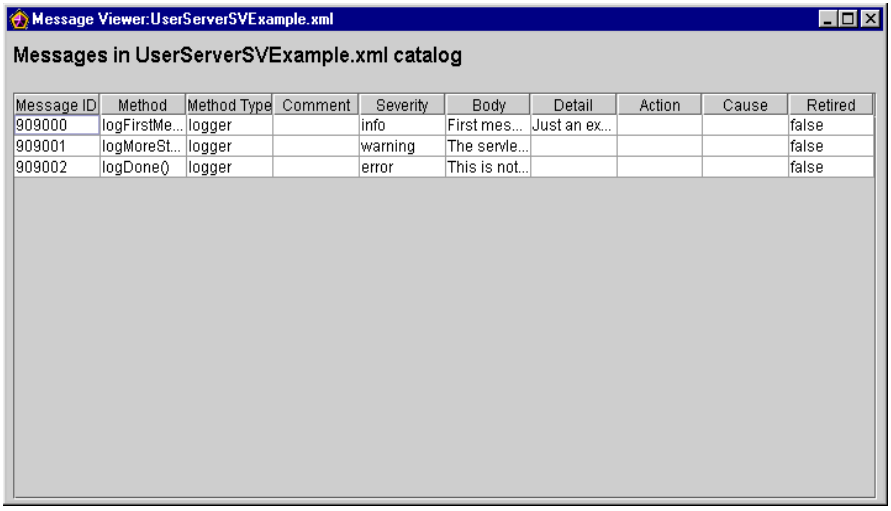
- Viewing All Messages in a Catalog
- Viewing All Messages in Several Catalogs
- Choosing a Message to Edit from the Message Viewer

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.
3. Select the **All messages** option. 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



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 3-4 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 909001

Editing an Existing Message

To edit an existing message:

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.

Retiring and Unretiring Messages

You can retire and unretire messages in the main Message Editor window. Retiring a message does not mean that the message is deleted from the master catalog. It simply means it is hidden from user view. This feature is useful for removing obsolete messages. If you need to bring a retired message back into view, you can unretire it.

To retire or unretire a message, complete the following procedure:

1. Find the message you want to retire or unretire, either by using the Search dialog as described in Finding a Log Message and Finding a Simple Text Message.
2. In the main Message Editor window, toggle the **Retired message** option on or off by clicking the checkmark box.
3. Click **Update**.

CHAPTER 4

Using the BEA WebLogic Server Internationalization Utilities

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

WebLogic Server Internationalization Utilities

WebLogic Server provides three internationalization utilities:

- `weblogic.i18ngen Utility`—Message catalog parser. Use this utility to validate and generate classes used for localizing text in log messages. For more information, see "weblogic.i18ngen Utility" on page 4-3.
- `weblogic.l10ngen Utility`—Locale-specific message catalog parser. Use this utility to process locale-specific catalogs. For more information, see "weblogic.l10ngen Utility" on page 4-6.
- `weblogic.GetMessage Utility`—Utility that lists installed log messages. Use this utility to generate a list of installed log messages or display a message. For more information, see "weblogic.GetMessage Utility" on page 4-8.

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.

Note: Use escapes to embed newlines (for example `'\n'`). These are stored and result in newlines when printed.

WebLogic Server Internationalization and Localization

Use the `weblogic.i18ngen` utility to validate message catalogs and create the necessary runtime classes for producing localized messages. The `weblogic.l10ngen` 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 configure the applications with `Logger` and `TextFormatter` classes generated from the `weblogic.i18ngen` utility.

Refer to "weblogic.i18ngen Utility" on page 4-3 for detailed information about the `weblogic.i18ngen` utility. For more information on `Logger` and `TextFormatter` classes, refer to Appendix C, "TextFormatter Class Reference for BEA WebLogic Server," and Appendix D, "Logger Class Reference for BEA WebLogic Server."

The generated `Logger` classes are used for logging purposes instead of the traditional method of writing English text to a log. For example, `weblogic.i18ngen` generates a class `xyzLogger` in the appropriate package for the catalog `xyz.xml`. For example, for the `MyUtilLog.xml` catalog, the class `programs.utils.MyUtilLogger.class` wouldbe generated. For each log message defined in the catalog, this class contains static public methods as defined by the `method` attributes.

`TextFormatter` classes are generated for each simple message catalog. These classes include 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.

weblogic.i18ngen Utility

Use the `weblogic.i18ngen` utility to parse message catalogs (XML files) to produce `Logger` and `TextFormatter` classes used for localizing the text in log messages. The utility creates or updates the following properties file (which is used to load the message id lookup class hashtable `weblogic.i18n.L10nLookup`):

`targetdirectory/i18n_user.properties`

The `weblogic.i18ngen` utility also creates or updates the `i18n_user.properties` file. Any errors, warnings, or informational messages are sent to `stderr`.

In order for user catalogs to be recognized, the `i18n_user.properties` file must reside in a directory identified in the WebLogic classpath.

For example:

```
targetdirectory/i18n_user.properties
```

BEA recommends that the `i18n_user.properties` file reside in the server classpath. If the `i18n_user.properties` file is in `targetdirectory`, then `targetdirectory` should be in the server classpath.

Syntax

```
java weblogic.i18ngen [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

Option	Definition
-build	Generates all necessary files and compiles them. The -build option combines the -i18n, -l10n, -keepgenerated, and -compile options.
`-d targetdirectory`	Specifies the root directory to which generated Java source files are targeted. User catalog properties are placed in `i18n_user.properties`, relative to the designated target directory. Files are placed in appropriate directories based on the `i18n_package` and `l10n_package` values in the corresponding message catalog. The default target directory is the current directory. This directory is created as necessary. If this argument is omitted, all classes are generated in the current directory, without regard to any class hierarchy described in the message catalog.
`-n`	Parse and validate, but do not generate classes.

`-keepgenerated`	Keep generated Java source (located in the same directory as the class files).
`-ignore`	Ignore errors.
`-i18n`	Generates internationalizers (for example, `Loggers` and `TextFormatters`). `i18ngen -i18n` creates the internationalizer source (for example, `*Logger.java`) that supports the logging of internationalized messages.
`-l10n`	Generates localizers (for example, `LogLocalizers` and `TextLocalizers`). `i18ngen -l10n` creates the localizer source (resource bundles) that provide access to each message defined in the message catalog. These classes are used by localization utilities to localize messages.
`-compile`	Compiles generated Java files using the current CLASSPATH. The resulting classes are placed in the directory identified by the -d option. The resulting classes are placed in the same directory as the source. Errors detected during compilation generally result in no class files or properties file being created. `i18ngen` exits with a bad exit status.
`-nobuild`	Parse and validate only.
-debug	Debugging mode.
-dates	Causes `weblogic.i18ngen` to update message timestamps in the catalog. If the catalog is writeable and timestamps have been updated, the catalog is rewritten.
`files`	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 `msgcat.dtd` syntax. `weblogic.i18ngen` prints the fully-qualified list of names (Java source) to the `stdout` log for those files actually generated.

weblogic.l10ngen Utility

The `weblogic.l10ngen` utility generates property resources for localizations of message catalogs named in the filelist. The file list identifies the top-level catalogs, not translated catalogs.

Similarly the *target* directory (`-d` option) identifies the same *target* directory where the default localizations reside. For example, if the default catalogs are located in `$SRC/weblogic/msgcat` and the generated resources are to be placed in `$CLASSESDIR`, the appropriate `l10ngen` invocation would be:

```
java weblogic.i18n.tools.l10ngen -d $CLASSESDIR $SRC/weblogic/msgcat
```

This command generates localized resources for all locales defined in the `weblogic/msgcat` subdirectories.

Syntax

```
java weblogic.l10ngen [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

Option	Definition
`-d` *target*	Directory in which to place properties. Default is the current directory.
`-language` *code*	Language code. Default is `all`.
`-country` *code*	Country code. Default is `all`.
`-variant` *code*	Variant code. Default is `all`.
-filelist	Specifies the message catalogs for which you want to generate properties files. You must specify top-level catalogs that the Message Editor creates; you do not specify locale-specific catalogs that the Message Localizer creates. Usually this is the same set of source files or source directories that you specified in the `i18ngen` command.

Message Catalog Localization

Catalog subdirectories are named after lowercase, two-letter ISO 639 language codes (for example, `ja` for Japanese and `fr` for French). You can find supported language codes in the `java.util.Locale` javadoc.

Variations to language codes are achievable through the use of uppercase, two-letter ISO 3166 country codes and variants, each of which are subordinate to the language code. The generic syntax is `lang/country/variant`.

For example, `zh` is the language code for Chinese. `CN` is a country code for simplified Chinese, whereas `TW` is the country code for traditional Chinese. Therefore `zh`/`CN` and `zh`/`TW` are two distinct locales for Chinese.

Variants are of use when, for instance, there is a functional difference in platform vendor handling of specific locales. Examples of vendor variants are WIN, MAC and POSIX. There may actually be two variants used to further qualify the locale. In this case, the variants are separated with an underscore (for example, `Traditional_Mac` vs. `Modern_MAC`).

Note: Language, country and variants are all case sensitive.

A fully-qualified locale would look like `zh`/`TW`/`WIN`, identifying traditional Chinese on a Win32 platform.

Message catalogs to support the above locale would involve the following files:

- `/*.xml`—default catalogs
- `/zh/*.xml`—Chinese localizations
- `/zh/TW/*.xml`—Traditional Chinese localizations
- `/zh/TW/WIN/*.xml`—Traditional Chinese localizations for Win32 code sets

Specific localizations do not need to cover all messages defined in parent localizations.

Examples

1. To generate localization properties for all locales:

   ```
   java weblogic.i18ntools.l10ngen -d $CLASSESEDIR catalogdirectory
   ```

2. To generate localization properties for all traditional Chinese locales:

   ```
   java weblogic.i18ntools.l10ngen -d $CLASSESEDIR -language zh -country TW
   catalogdirectory
   ```

3. To generate localization properties for all Chinese locales:

   ```
   java weblogic.i18ntools.l10ngen -d $CLASSESEDIR -language zh
   catalogdirectory
   ```

4. To generate localization properties for the JMS catalog in all locales:

   ```
   java weblogic.i18ntools.l10ngen -d $CLASSESEDIR catalogdirectory
   ```

Note: Example 2 is a subset of example 3. All Chinese (`zh`) would include any country designations (for example, `TW`) and variants.

`weblogic.l10ngen` does not validate the locale designators (language, country, variant).

weblogic.GetMessage Utility

The `weblogic.GetMessage` utility replaces the CatInfo utility provided with earlier releases of WebLogic Server. This utility displays message content but can also be used to list all or some subset of installed messages. By default (no options), `weblogic.GetMessage` prints a usage statement.

Syntax

```
java weblogic.GetMessage [options]
```

Options

Note: All options may be abbreviated to a single character except `-verbose`.

Option	Definition
`-id` *nnnnnn*	where *nnnnnn* represents the message ID. The `-id` option is used to specify a particular message.
`-subsystem` *identifier*	The subsystem identifier. The `-subsystem` option prints only those messages that match the specified subsystem.

Option	Definition
`-nodetail`	Requests a non-detailed listing, and only outputs the message body of a message. By default, a detailed listing is output, which includes severity, subsystem, message detail, cause, and action information.
`-verbose`	Requests more detail on the listing. The `-verbose` option also prints packaging, stacktrace option, severity, subsystem, message detail, cause, and action information.
`-lang code`	The language to use. For example, `en` for English.
`-country code`	The country code to use. For example, `US` for United States.
`-variant code`	The variant designator for locale.
`-help`	Provides usage information.
`-retired`	Lists all retired messages. Retired messages are not displayed unless this option is used. Only the subsystem and id's of such messages are listed.

If no arguments are provided, `weblogic.GetMessage` outputs a usage message, equivalent to `-help`.

APPENDIX A

Localizer Class Reference for BEA WebLogic Server

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

Note: This information on `Localizer` class methods is provided as reference for advanced users. Normally, you do not need to use these interfaces directly. Instead, you would typically use the generated methods in the catalogs.

About Localizer Classes

`Localizers` are classes that are used by applications and server code to localize text for output. The `weblogic.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`** for log message catalogs and **`TextLocalizer`** for simple text catalogs. A **`Localizer`** class for the catalog `ejb.xml` is `ejb`**`LogLocalizer`**.

Localizer Methods

Localizers are `PropertyResourceBundle` objects. Four additional methods are provided to simplify the access of the localization data in the `Localizer`. These methods are described in Table A-1.

These methods are not part of the `Localizer`. Rather, they are part of the `Localizer` class. The `Localizer` class is used by the `Logger` and `TextFormatter` classes to extract data out of the `Localizer`. Each `Localizer` has an associated `Localizer` class that is obtained through `L10nLookup`, the `Localizer` lookup object.

Table A-1 Methods for Localization Data Access

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

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, which is a property class extension that is loaded at system startup from the property file:

i18n_user.properties

This property file is created by `weblogic.i18ngen` and is included in the WebLogic Server distribution. When you start up a user application, any `i18n_user.properties` files in its classpath are also loaded into `L10nLookup`.

Properties in the lookup (`i18n_user.properties`) 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`**

Loggable Object Reference for BEA WebLogic Server

The following sections provide reference information for loggable objects:

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 To Use Loggable Objects

To create a `Logger` class 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 B-1.

Listing B-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/wls810/dtd/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 `weblogic.i18ngen` utility, a `Logger` class is created for this catalog with the following two methods:

- `logIOError` (throwable)—logs the message
- `logIOErrorLoggable` (throwable)—returns a loggable object

The loggable object can be used as shown in Listing B-2.

Listing B-2 Example of Use of Loggable Object

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

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

APPENDIX C

TextFormatter Class Reference for BEA WebLogic Server

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

About TextFormatter Classes

`TextFormatter` classes are generated by `weblogic.i18ngen` from simple message catalogs. These classes provide methods for generating localized versions of message text at runtime.

Example of an Application Using a TextFormatter Class

The following is an example of a `Hello_World` application, its simple message catalog, and the `TextFormatter` class generated for the catalog.

Listing C-1 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/wls810/dtd/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>
```

The following is an example of an application using the HelloWorld catalog. The example shows various ways of internationalizing an application using simple message catalogs.

Listing C-2 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
 * using the catalog editing utility, weblogic.i18ntools.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/i18n/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
   */
  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()));
  }
}
```

The following listing shows an example of the generated `TextFormatter` for the HelloWorld catalog.

Listing C-3 Example of Generated TextFormatter Class for the HelloWorld Catalog

```
package examples.i18n.simple;


import java.text.MessageFormat;

import java.text.DateFormat;

import java.util.Date;

import java.util.Locale;

import weblogic.i18n.Localizer;

import weblogic.i18ntools.L10nLookup;

/**

* Copyright (c) 2002 by BEA Inc. All Rights Reserved.

* @exclude

*/

public class HelloWorldTextFormatter {

  private Localizer l10n;

  private boolean format=false;

  // constructors
```

```
  public HelloWorldTextFormatter() {
  l10n = L10nLookup.getLocalizer(Locale.getDefault(),
"examples.i18n.simple.HelloWorldTextLocalizer");
  }
  public HelloWorldTextFormatter(Locale l) {
  l10n =
L10nLookup.getLocalizer(l,"examples.i18n.simple.HelloWorldTextLocalizer");
  }
  public static HelloWorldTextFormatter getInstance() {
    return new HelloWorldTextFormatter();
  }
  public static HelloWorldTextFormatter getInstance(Locale l) {
    return new HelloWorldTextFormatter(l);
  }
  public void setExtendedFormat(boolean fmt) {
    format = fmt;
  }
  public boolean getExtendedFormat() { return format;
  /**
   * Hello World!
   */
  public String helloWorld()  {
    String fmt  = "";
    String id = "HELLO_WORLD" ;
    String subsystem = "I18N" ;
    Object [] args = {  };
    String output =  MessageFormat.format(l10n.get(id) , args);
    if (getExtendedFormat()) {
```

```
    DateFormat dformat = DateFormat.getDateTimeInstance(DateFormat.MEDIUM,
DateFormat.LONG);
     fmt = "<"+dformat.format(new Date())+"><"+subsystem+"><"+id+"> ";
     }
     return fmt+output;
     }

 /**
   * Hello again
   */
  public String helloAgain()  {
     String fmt  = "";
     String id = "HELLO_AGAIN" ;
     String subsystem = "I18N" ;
     Object [] args = {  };
     String output =  MessageFormat.format(l10n.get(id) , args);
     if (getExtendedFormat()) {
     DateFormat dformat = DateFormat.getDateTimeInstance(DateFormat.MEDIUM,
DateFormat.LONG);
       fmt = "<"+dformat.format(new Date())+"><"+subsystem+"><"+id+">
     ";
     }
     return fmt+output;
     }
 /**
   * This is hello number {0,number}.
   */
  public String nthHello(int arg0)  {
```

```
    String fmt  = "";
    String id = "NTH_HELLO" ;
    String subsystem = "I18N" ;
    Object [] args = { new Integer(arg0) };
    String output =  MessageFormat.format(l10n.get(id) , args);
    if (getExtendedFormat()) {
    DateFormat dformat = DateFormat.getDateTimeInstance(DateFormat.MEDIUM,
DateFormat.LONG);
     fmt = "<"+dformat.format(new Date())+"><"+subsystem+"><"+id+">
    ";
    }
    return fmt+output;
    }
 /**
   * Catalog version: {0}
   */
   public String version(String arg0)  {
    String fmt  = "";
    String id = "VERSION" ;
    String subsystem = "I18N" ;
    Object [] args = { arg0 };
    String output =  MessageFormat.format(l10n.get(id) , args);
    if (getExtendedFormat()) {
   DateFormat dformat = DateFormat.getDateTimeInstance(DateFormat.MEDIUM,
DateFormat.LONG);
    fmt = "<"+dformat.format(new Date())+"><"+subsystem+"><"+id+">
    ";
    }
```

```
    return fmt+output;
    }
 /**
   * I18n Package: {0}
   */
  public String i18nPackage(String arg0)  {
    String fmt  = "";
    String id = "I18N_PACKAGE" ;
    String subsystem = "I18N" ;
    Object [] args = { arg0 };
    String output =  MessageFormat.format(l10n.get(id) , args);
    if (getExtendedFormat()) {
    DateFormat dformat =
DateFormat.getDateTimeInstance(DateFormat.MEDIUM,DateFormat.LONG)
    ;
    fmt = "<"+dformat.format(new Date())+"><"+subsystem+"><"+id+">
    ";
    }
    return fmt+output;
    }
 /**
   * L10n Package: {0}
   */
  public String l10nPackage(String arg0)  {
    String fmt  = "";
    String id = "L10N_PACKAGE" ;
    String subsystem = "I18N" ;
    Object [] args = { arg0 };
```

```
    String output =  MessageFormat.format(l10n.get(id) , args);
    if (getExtendedFormat()) {
    DateFormat dformat = DateFormat.getDateTimeInstance(DateFormat.MEDIUM,
DateFormat.LONG);
    fmt = "<"+dformat.format(new Date())+"><"+subsystem+"><"+id+">
    ";
    }
    return fmt+output;
    }
 /**
   * Catalog subsystem: {0}
   */
    public String subSystem(String arg0)  {
    String fmt  = "";
    String id = "SUBSYSTEM" ;
    String subsystem = "I18N" ;
    Object [] args = { arg0 };
    String output =  MessageFormat.format(l10n.get(id) , args);
    if (getExtendedFormat()) {
    DateFormat dformat = DateFormat.getDateTimeInstance(DateFormat.MEDIUM,
DateFormat.LONG);
    fmt = "<"+dformat.format(new Date())+"><"+subsystem+"><"+id+">
    ";
    }
    return fmt+output;
    }
    }
```

Logger Class Reference for BEA WebLogic Server

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

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.

For more information, refer to Appendix B, "Loggable Object Reference for BEA WebLogic Server."

Example of a Generated Logger Class

Listing D-1 contains an example of a generated logger class.

Listing D-1 Example of Generated Logger Class

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE message_catalog PUBLIC "weblogic-message-catalog-dtd"
"http://www.bea.com/servers/wls810/dtd/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 D-2 shows the corresponding Java source generated by `weblogic.i18ngen`.

Listing D-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
   *
   * 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 D-3 shows an example application that uses the `weblogic.i18nLog` (internationalized (I18n) logging interfaces). The example logs an informational message.

Listing D-3 Example of Application Using i18nLog

```
package examples.i18n.logging;

import java.util.Locale;

import weblogic.i18n.Localizer;
import weblogic.i18ntools.L10nLookup;
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.
 */
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 appplications 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)));

    }
}
```

Index

M

N

P

R

S

U