BEA WebLogic Server

Internationalization Guide

BEA WebLogic Server 6.1
Document Date: June 24, 2002

Internationalization Guide

Part Number	Document Date	Software Version
N/A	June 24, 2002	BEA WebLogic Server Version 6.1

Contents

About This Document

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

The document is organized as follows:

- Chapter 1, “Overview of Internationalization for WebLogic Server,” an overview of the processes required for internationalization and localization.
- Chapter 2, “Using Message Catalogs with BEA WebLogic Server,” a description of a message catalog types and message definitions, elements, and arguments.
- Chapter 3, “Using the BEA WebLogic Server Internationalization and Localization Interfaces,” contains an overview of how the interfaces are used.
- Chapter 4, “Using the BEA WebLogic Server Internationalization Tools and Utilities,” how to use the internationalization tools that are included with WLS.
- Chapter 5, “Instrumenting Your Code for Localization,” how to use code instrumentation to create a new catalog, and to log, create or modify a message.
- Appendix A, “Localizer Class Reference for BEA WebLogic Server,” <add description of appendix>.
- Appendix B, “Logger Class Reference for BEA WebLogic Server,” <add description of appendix> .

Audience

This document is written for application developers who must internationalize or localize the message catalogs included in the WLS distribution for locale-specific administration and management. It is assumed that readers have a familiarity with the WebLogic Server platform and Java programming.

e-docs Web Site

BEA product documentation is available on the BEA corporate Web site. From the BEA Home page, click on Product Documentation or go directly to the WebLogic Server Product Documentation page at http://e-docs.bea.com/wls/docs60.

How to Print the Document

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

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

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

Related Information

The following WebLogic Server documents contain information that is relevant to using the idltojava compiler and understanding how to implement Java CORBA applications in the WLE system.

For more information in general about Java IDL and Java CORBA applications, refer to the following sources.

- The OMG Web Site at http://www.omg.org/
- The Sun Microsystems, Inc. Java site at http://java.sun.com/

For more information about CORBA and distributed object computing, transaction processing, and Java, refer to the Bibliography at http://edocs.beasys.com/.

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 your 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.

`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. *Examples*: LPT1 BEA_HOME OR
`{ }`	A set of choices in a syntax line.

Convention	Usage

`[ ]`	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
. . .	Indicates the omission of items from a code example or from a syntax line.
Convention	**Usage**
Ctrl+Tab	Keys you press simultaneously.

CHAPTER

1 Overview of Internationalization for 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. The information covered in this guide only addresses the presentation of textual data.

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.

This section includes the following topics:

- Understanding Log Messages
- Understanding Localization
- Java Interfaces for Internationalization

Understanding Log Messages

To create an internationalized message, all message strings must be externalized in a message catalog so that they can be easily converted to multiple locales without changing or recompiling the code. The application code supplies any run-time values to the logging methods, which merge the code with the message strings in the catalog per the current locale, and prints a localized message in the log files.

Within the message catalog file, each 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 find it easily.

The following steps are involved in creating an internationalized message:

1. Define the message in a message catalog. For details, see "Using the BEA WebLogic Server Internationalization Tools and Utilities."

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

2. Run `i18ngen`, which validates the message catalog and generates a `Logger` (a Java class) for each catalog. The generated class has a method for each message, according to information specified in the message catalog entry.

3. When a particular message needs to be logged, the application code must invoke the relevant generated method from its `Logger` and provide the run-time values as arguments.

4. The `Logger` method combines the run-time values and the message strings in the catalog to create localized entries in the WebLogic Server log files.

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

Understanding Localization

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.

WebLogic Server internationalization supports localization of two types of data:

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

Java Interfaces for Internationalization

Users of the Java internationalization interfaces should be familiar with the following interfaces included in the Java Developer's Kit (JDK):

`java.util.Locale`	Represents a specific geographical, political, or cultural region.
`java.util.ResourceBundle`	Containers for locale-specific objects.
`java.text.MessageFormat`	A means to produce concatenated messages in a language-neutral way.

Internationalization catalogs reside as XML files in the directory

```
weblogic/msgcat
```

Localization catalogs reside as XML files in subdirectories to `weblogic/msgcat`. (see Catalog Hierarchy). These XML catalogs are compiled into classes during the build process; the methods of the resulting classes are the objects used to log messages at runtime.

The message catalogs are XML files that contain a description of a collection of text messages, each indexed by a unique idenifier. Catalogs for the base locale are defined by the document type definition (DTD) in `msgcat.dtd`. Catalogs which provide different localizations of the base catalogs are defined in `msgcat` subdirectories named for the locale (for example, `weblogic/msgcat/ja` for Japan), and use `l10n_msgcat.dtd`.

CHAPTER

2 Using Message Catalogs with BEA WebLogic Server

All internationalized text is defined in message catalogs, each of which defines a collection of log messages or simple text.

A message catalog is an XML file that is defined in a document type description (DTD), and defines text messages. Each catalog defines the log messages or simple text used by some subset of the system. Users may choose to create a single log message catalog for all their logging requirements, or to create smaller catalogs based on a subsystem or Java package. We recommend the latter method. Using multiple subsystems allows you to focus on specific portions of the log during viewing.

For simple text catalogs, the recommended approach is to create a single catalog for each utility being internationalized. Developers can create site-specific message catalogs using the tools described in "Using the BEA WebLogic Server Internationalization Tools and Utilities."

All messages and exceptions must be defined in a default, top-level catalog. WLS is distributed with a collection of catalogs located in `/weblogic/msgcat` within the `weblogic.jar` file.

This section presents information about the following topics related to message catalogs:

- Message Catalog Types
- Including Message Arguments

- Choosing Names for Message Catalogs
- Message Catalog Hierarchy
- Message Catalog Formats

Message Catalog Types

DTDs that describe the syntax of the message catalogs are also stored in the `weblogic/msgcat` directory. Two DTDs are included in the WLS distribution:

- `msgcat.dtd`–Describes the syntax of top-level, default catalogs. These catalogs include information that is not in the locale-specific catalogs, such as message severity.
- `l10n_msgcat.dtd`—Describes the syntax of locale-specific catalogs.

In this directory you will also find templates that can be used to create top-level and locale-specific message catalogs.

Including Message Arguments

All of the sections of a log message can include message arguments, as described by `java.text.MessageFormat`. A message can support up to 10 arguments, numbered 0-9. Any subset of these arguments can be included in any of the text sections of the message definition. Message arguments are inserted into a message definition during development, and are replaced by the appropriate message content at run time when the message is logged.

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

```
<messagebody>Unable to open file, {0}.</messagebody>
    <messagedetail>
```

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

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

```
-method=“logNoFile(String name, String path)”
```

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

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

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

Choosing Names for Message Catalogs

The name of the message catalog file (without the `.xml` extension) is used to generate Localizer and Logger class names; it should be chosen carefully. For example, a name for a message catalog should:

- Not conflict with any names of existing classes in the target package
- Not contain characters that are not allowed in class names
- Follow class naming standards

For example, the resulting class names for a catalog named `Xyz.xml` are `XyzLogLocalizer` and `XyzLogger`.

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

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.

Message Catalog Hierarchy

Message catalogs support multiple locales or languages. For a specific message catalog there is exactly one default version, known as the top-level catalog, and one catalog for each additional supported locale. For example, you might have a top-level catalog named `mycat.xml`, and a Japanese translation of it called `../ja/mycat.xml`. Typically the top-level catalog is English, but English is not required for any catalogs except the installed WLS catalogs.

Locale designations (for example, `ja`) also have a hierarchy as defined in the `java.util.Locale` documentation. Briefly, a locale can include a language, country and variant. Language is the most standard. Language can be extended with a country code. For instance, `en/US`, indicates American English. The name of the associated catalog would be `../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.

Message Catalog Formats

The catalog format for top-level and locale-specific catalog files is slightly different; locale-specific catalogs only provide translations of text defined in the top-level version. Furthermore log message catalogs are defined differently than simple text catalogs. The differences are made evident in the descriptions that follow.

- Elements of a Log Message Catalog
- Format of a Simple Text Message Catalog

- Format of a Locale-Specific Catalog

Elements of a Log Message Catalog

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

- `message_catalog`
- `log_message`
- Other Log Message Catalog Elements

message_catalog

The following table describes the attributes that can be defined for the `message_catalog` element.

Attribute...	**Description...**
`i18n_package`	Optional. Java package in which to place generated Logger classes for this catalog. The classes are named after the catalog file name. For exampale, for a catalog using `mycat.xml`, a generated logger class called `*i18n_package*.mycatLogger.class`. The default is `weblogic.i18n`.
`l10n_package`	Optional. Java package to place generated `LogLocalizer` properties for this catalog. Classes are named after the catalog file name. For example, for a catalog called `mycat.xml` a properties file called `*l10n_package*.mycatLogLocalizer.properties` is generated. The default is `weblogic.i18n`
`subsystem`	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.
`version`	Required. Specifies the version of the `msgcat.dtd` being used. The usage is `*n.n*`, for example, `version="1.0"`
`baseid`	Optional. Specifies the lowest message ID used in this catalog. The syntax is one to six decimal units. Default is `000000` for WLS catalogs, `500000` for user catalogs.

Attribute...	Description...
`endid`	Optional. Specifies the highest message ID used in this catalog. The syntax is one to six decimal units. Default is 499999 for WLS catalogs, 999999 for user catalogs.

log_message

The following table describes the attributes that can be defined for the `log_message` element.

Attribute...	Description...
`messageid`	Required. Unique identifier for this log message. Uniqueness should extend across all catalogs. Value must be in range defined by baseid and endid attributes defined above. This is a child element of `message_catalog`.
`datelastchanged`	Optional. Date/time stamp useful for managing modifications to this message. The date will be supplied by utilities which run on the catalogs The syntax is `Long.toString(new Date().getTime());`
`severity`	Required. The severity of the log message. Must be one of the following: "debug","info", "warning", "notice", "error", "critical", "alert", or "emergency". User catalogs may only use "debug", "info", "warning", and "error".
`stacktrace`	Optional. Indicates whether to generate a stack trace for Throwable arguments. Possible values are `true` or `false`. Default is `true`. When the value is `true` a trace is generated. The syntax is: `stacktrace="true"`

Attribute...	**Description...**
`method`	Required. Method signature for logging this message. Two methods are actually provided: the one specified here and a similar one with an additional Throwable argument. The syntax is the standard Java method signature, less qualifiers, semicolon, and extensions. Arguments 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, they should have a useful `toString()` method. Argument can be any valid name, but should follow the convention of `argn` where `n` is 0 thru 9. There can be no more than ten (10) arguments. For each `argn` there should be at least one corresponding placeholder in the text elements described in "Other Log Message Catalog Elements." Placeholders are of the form `{n}`, `{n,`*`number`*`}` or `{n,`*`date`*`}`.

Other Log Message Catalog Elements

Element...	**Description...**
`messagebody`	Required. A string containing a short description for this message. This element may contain zero or more placeholders, `{n}`, which will be replaced by the appropriate argument when the log message is localized. This is a child element of `log_message`.
`messagedetail`	Required. A string containing a detailed description of the event. This element may contain zero or more placeholders, `{n}`, which will be replaced by the appropriate argument when the log message is localized. This is a child element of `log_message`.
`cause`	Required. A string describing the root cause of the problem. This element may contain zero or more placeholders, `{n}`, which will be replaced by the appropriate argument when the log message is localized. This is a child element of `log_message`.
`action`	Required. A string describing the recommended resolution. This element may contain zero or more placeholders, {n}, which will be replaced by the appropriate argument whenn the log message is localized. This is a child element of `log_message`.

Log Message Catalog Syntax

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

```
<?xml version="1.0"?>
<!DOCTYPE message_catalog PUBLIC "weblogic-message-catalog-dtd"
"http://www.bea.com/servers/wls610/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>
```

Format of a Simple Text Message Catalog

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

- message_catalog
- message
- messagebody

message_catalog

The following table describes the attributes that can be defined for the `message_catalog` element

Attribute...	**Description...**
`l10n_package`	Optional. Java package to place generated LogLocalizer properties for this catalog. The classes are named after the catalog file name. `mycat.xml` would generate the properties file `*l10n_package*.mycatLogLocalizer.properties`. The default is "weblogic.i18n"
`subsystem`	Required. An acronym identifying the associated subsystem for this catalog. The subsystem is included in the error log and used for message isolation purposes.
`version`	Required. Specifies the version of the `msgcat.dtd` being used. Must be at least "1.0".

message

The following table describes the attributes that can be defined for the `message` element.

Attribute...	**Description...**
`messageid`	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`	Optional. Date/time stamp useful for managing modifications to this message.

Attribute...	Description...

`method`	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

The text associated with the message.

Use Required. This element may contain zero or more placeholders, `{n}`, which will be replaced by the appropriate arguments when the log message is localized. `messagebody` is a child element of message.

Simple Text Catalog Example

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

```
<?xml version="1.0"?>
<!DOCTYPE message_catalog PUBLIC "weblogic-message-catalog-dtd"
    "http://www.bea.com/servers/wls610/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>
```

Format 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 following elements and attributes are valid in locale-specific catalogs:

- `locale_message_catalog` element—Equivalent to `message_catalog` in a top-level definition
- `message` element—Equivalent to `message` in a top-level definition.
- `messageid` attribute—Governed by the same rules that apply to a top-level catalog. Must match a `messageid` in the associated top-level catalog.
- `messagebody`, `messagedetail`, `cause`, `action`—Governed by the same rules that apply to a top-level catalog.

For example, a French translation of the message in `MyUtilLabels.xml` is available in the following directory:

```
   .../msgcat/fr/MyUtilLabels.xml
```

The translated message appears as follows:

```
<?xml version="1.0"?>
<!DOCTYPE message_catalog PUBLIC
   "weblogic-locale-message-catalog-dtd"
   "http://www.bea.com/servers/wls610/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 UTF-8 characters, and haveappropriate keyboard mappings installed. The `MessageLocalizer` utility is an example of such a tool.

3 Using the BEA WebLogic Server Internationalization and Localization Interfaces

This section contains information about the interfaces used for internationalization and localization.

- Using the Internationalization Interfaces
- Using the Localization Interfaces

Using the Internationalization Interfaces

Internationalization of simple text-based utilities is achieved by specifying that those utilities must use Localizers to access text data. For logging purposes, however, the generated Logger classes must be used instead of the traditional method of writing English text to a log. For example, `i18ngen` generates a class `xyzLogger` in the appropriate package for the catalog `xyz.xml`.

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

Using the Localization Interfaces

TextFormatter classes are generated for each simple message catalog. These classes include static methods for accessing localized and formatted text from the catalog. They are convenience classes that handle the interface with the message body, placeholders, and MessageFormat. The formatting methods are specified via the method attribute in each message definition. For example, if a message definition included the attribute `method=getErrorNumber(int err)` then the resulting TextFormatter class would apprear as follows:

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

Use of the above method would resemble the following code example:

```
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.

Two constructors are provided. The default constructor results in the use of the default locale for the Java Virtual Machine (JVM). The second constructor allows specification of a different locale without changing the locale for the entire JVM.

CHAPTER

4 Using the BEA WebLogic Server Internationalization Tools and Utilities

This section includes information about the following topics:

- Using the Message Editor
- Using the Internationalization Utilities

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

Using the Message Editor

The Message Editor is a GUI for editing XML message catalogs. It creates, reads, and writes XML catalogs. Optionally, you can also edit the XML catalogs directly, in a text editor. Message Editor allows you to perform the following tasks:

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

The catalog current being created or used in the Message Editor is referred to as the *context catalog*.

The catalog parser recognizes special character references and converts them to the intended character, as shown in Table 4-1. The Message Editor will recognize the special characters and write them back using the character references. The parser recognizes no other character references.

Table 4-1 Special Character Reference

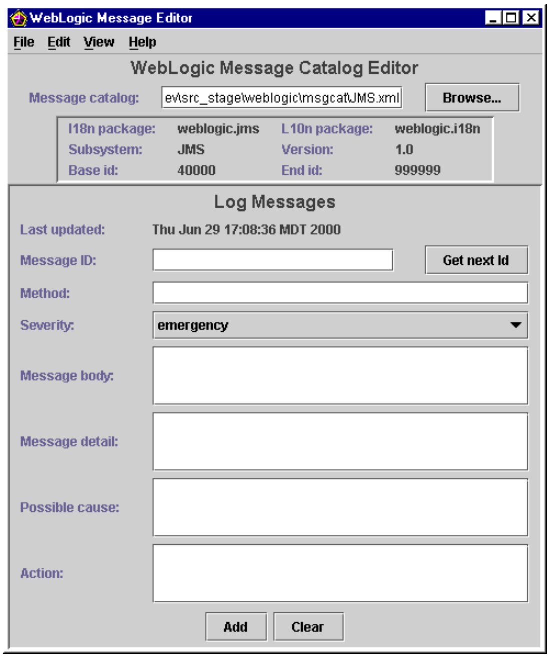
Character	Character Code
`@`	`&amp;`
`<`	`&lt;`
`>`	`&gt;`
`‘`	`&apos;`
`“`	`&quot;`

Starting the Message Editor

To start the Message Editor, type:

```
java weblogic.MsgEditor
```

The main Message Editor for a log message is displayed as follows:



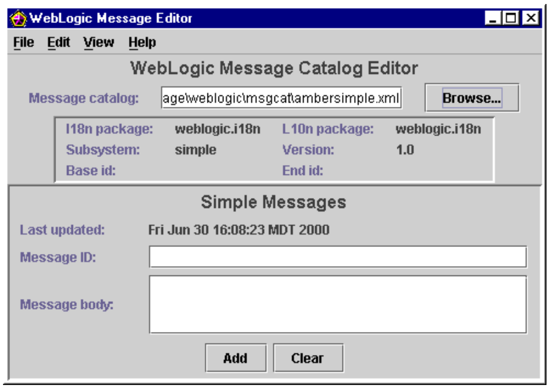
WebLogic Message Catalog Editor

Message catalog:	ev\src_stage\weblogic\msgcat\JMS.xml		
I18n package:	weblogic.jms	L10n package:	weblogic.i18n
Subsystem:	JMS	Version:	1.0
Base id:	40000	End id:	999999

Log Messages

Last updated:	Thu Jun 29 17:08:36 MDT 2000
Message ID:	
Method:	
Severity:	emergency
Message body:	
Message detail:	
Possible cause:	
Action:	

When displaying a simple message catalog in the Message Editor, it will appear as follows:



WebLogic Message Catalog Editor

Message catalog:	age\weblogic\msgcat\ambersimple.xml		
I18n package:	weblogic.i18n	L10n package:	weblogic.i18n
Subsystem:	simple	Version:	1.0
Base id:		End id:	

Simple Messages

Last updated:	Fri Jun 30 16:08:23 MDT 2000
Message ID:	
Message body:	

Editing an ExistingCatalog

To edit an existing catalog:

- Enter the full pathname in the Message Catalog field, or click the Browse button and navigate to the existing catalog.

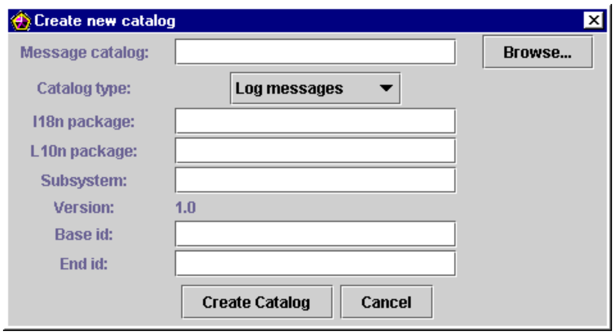
The catalogs can be found in the directory `$SRC/weblogic/msgcat`. Once the catalog has been located, the packages, subsystem, version and base id and end id (if any) for that catalog are displayed, and that catalog is the catalog context in which all other actions are performed. You are now ready to enter new messages into that catalog, to edit existing messages, to search for a message, or to view all messages in the catalog.

Creating a New Catalog

To create a new catalog, complete the following procedure:

1. Choose the **File** menu from the main menu bar.
2. Choose **New Catalog**.

   A “Create new catalog” dialog box is displayed, as shown here.


Create new catalog

Message catalog:
Catalog type: Log messages
I18n package:
L10n package:
Subsystem:
Version: 1.0
Base id:
End id:

3. Enter the full pathname in the **Message Catalog** field, or click the **Browse** button and navigate to the WebLogic catalog directory, `$SRC/weblogic/msgcat`.
4. Enter the name of the new catalog, which must include the `xml` extension.
5. Use the dropdown list to indicate whether your catalog is to be a **log message** catalog or a **simple text** message catalog.

If this is to be a log message catalog, the **Base ID** and **End ID** fields are displayed; if it is to be a simple text message catalog, those fields disappear.

6. Enter the remainder of the fields, and click **Create Catalog**.

   The dialog box disappears, and the catalog you just created is displayed as the context catalog in the main Message Editor window.

Entering a New Log Message

The entire catalog is written to disk immediately upon adding the message.

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

1. Enter a message ID, or click the **Get next ID** button to get the next numerical ID which is unique in the context catalog.
2. Enter the appropriate method, including parentheses and any arguments. For example:

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

3. Choose a severity from the list of possible levels.
4. Enter the message body, detail, cause, and action. Parameters are denoted by `{`*`n`*`}`. For example:

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

5. Click **Add**.

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

Entering a New Simple Text Message

The entire catalog is written to disk immediatly upon adding the message.

1. Enter an alphanumeric **ID** for the message.

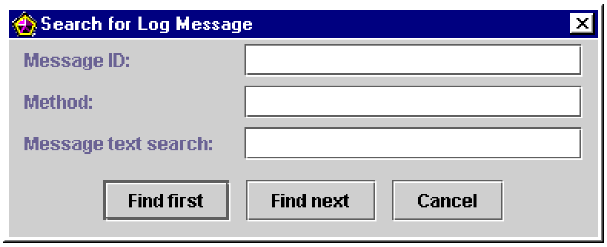
   This number must be unique within the catalog.

2. Enter the message body.
3. Click **Add.**

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

Finding a Log Message

1. Make sure that the context catalog is a log catalog.
2. Choose **Edit** from the main menu bar.
3. Choose **Search** to display the search dialog box.



4. Enter as much information as you need to search for and click **Find first** or **Find next**.

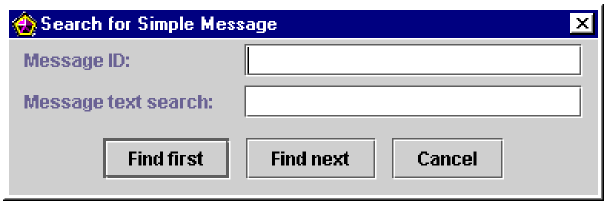
   The fields are strung together to find the message. The message ID and the method name must be complete, but the search for text does a partial match in any of the text fields.

If a matching message is found, it is displayed in the main Message Editor window, ready for editing.

Finding a Simple Text Message

Make sure that the context catalog is a simple text message catalog.

1. Choose **Edit** from the main menu bar.
2. Choose **Search** to display the search dialog box.



3. Enter as much information as you need to search for and click **Find first** or **Find next**.

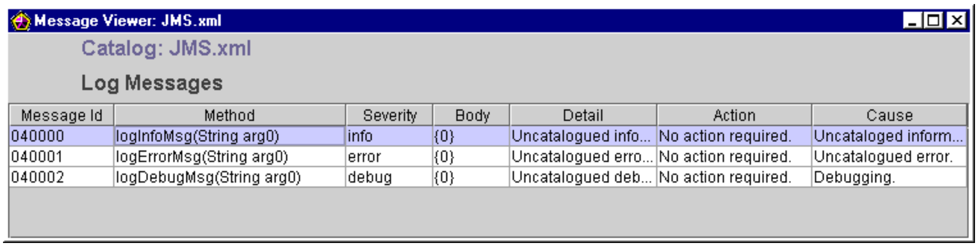
The fields are strung together to find the message. The message ID must be complete, but the search for text does a partial match.

If a matching message is found, it is displayed in the main Message Editor window, ready for editing.

Viewing All Messages in a Catalog

To view all the message in a catalog, choose **View** from the main menu bar.

All the messages for the current catalog context are displayed in a table in the Message Viewer window, as shown.



Message Id	Method	Severity	Body	Detail	Action	Cause
040000	logInfoMsg(String arg0)	info	{0}	Uncatalogued info...	No action required.	Uncataloged inform...
040001	logErrorMsg(String arg0)	error	{0}	Uncatalogued erro...	No action required.	Uncatalogued error.
040002	logDebugMsg(String arg0)	debug	{0}	Uncatalogued deb...	No action required.	Debugging.

Choosing a Message to Edit from the Catalog List

If you click on any row of displayed messaged in the Message Viewer, the designated row is selected, and the corresponding message appears in the main Message Editor window, ready for editing.

Viewing All Messages in Several Catalogs

If you view the messages from the current catalog context, and then change the context by browsing to a new catalog, the old view of the old catalog remains on the screen while you view the new catalog. Repeating this step allows you to view, at the same time, as many catalogs as you require (or can reasonably fit on your screen).

Choosing a Message to Edit from Any Catalog Shown

When you click on any row in any view the catalog context switches to the viewed catalog, and the message in that catalog is displayed in the Message Editor main window.

Editing an Existing Message

The entire catalog is written to disk immediately upon adding the message.

1. Find the message you want to edit, either by using the Search dialog box, or by clicking on a row in the message viewer.
2. Edit the fields you wish to change in the main Message Editor window.
3. Click **Update**.

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

Using the Internationalization Utilities

WebLogic Server provides three internationalization utilities:

- `i18ngen`—Message catalog parser
- `l10ngen`—Locale-specific message catalog parser

- CatInfo—Utility that lists installed log messages

i18ngen

The i18ngen utility parses message catalogs (XML files) to produce classes used for localizing the text in log messages. Used to compile the top-level message catalogs into Java classes and properties files.

Syntax

```
i18ngen [options] files
```

Options

Option	**Definition**
-d *targetdirectory*	Send generated Java files to this target. User catalog properties are placed in i18n_user.properties, relative to the target directory named in the -d designation.
-n	Parse and validate, but do not generate classes
-keepgenerated	Keep generated Java source
-ignore	Ignore errors
-i18n	Generate internationalizers (for example, Loggers)
-l10n	Generate Localizers (for example, LogLocalizers)
-compile	Compile generated source
-nobuild	Parse and validate only
files	Process the files and directories in this list of files.

The i18ngen utility creates or updates the i18n_user.properties file.

The option `i18ngen -i18n` creates the internationalizer source (for example, `*Logger.java`) that supports the logging of internationalized messages.

The option `i18ngen -l10n` creates the localizer source (property resource bundles) that provides access to each message defined in each message catalog. These properties are used by localization utilities to localize messages.

Errors detected during compilation generally result in no class files or properties file being created. `i18ngen` exits with a bad exit status.

The target directory (`-d` option) specifies the root directory in which generated source files are generated. Files are placed in appropriate directories based on the `i18n_package` and `l10n_package` values in the corresponding message catalog. The default target dir is the current directory.

The `-compile` option compiles generated Java files using the current `CLASSPATH`. The resulting classes are placed in the directory identified by the `-d` option.

`The i18ngen` utility processes all files listed on the command line. 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. `i18ngen` prints the fully-qualified list of names (Java source) to the `stdout` log for those files actually generated. Any errors, warnings, or informational messages are sent to `stderr`.

l10ngen

The `l10ngen` utility processes the locale-specific catalogs in directories that are subordinate to the top-level catalogs.

Syntax

```
java -classpath <l10n_Classpath> weblogic.i18ntools.l10ngen
[options] filelist
```

where `<l10n_Classpath>` should include `<WebLogic Home>/lib/weblogic.jar`

Options

Option	Definition
`-d` *`targetdirectory`*	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`*	Message catalog directories and files to process, relative to the current directory. Identifies top-level, not local-specific, directory.

CatInfo

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

Syntax

```
java weblogic.i18n.tools.CatInfo [options]
```

Options

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

Option	Definition
`-id` *`nnnnnn`*	where *`nnnnnn`* represents the message ID. The `-id` option is used to specify a particular message.

Option	**Definition**
`-subsystem` *`identifier`*	The subsystem identifier. The `-subsystem` option prints only those messages that match the specified subsystem.
`-detail`	Requests a detailed listing. The `-detail` option also requests version, severity, subsystem, message detail, cause, and action information.
`-help`	Provides usage information.

To export the detailed list of messages to a file:

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

CHAPTER

5 Instrumenting Your Code for Localization

This section discusses the following types of code instrumentation:

- Logging a Message
- Creating a New Catalog
- Creating New Messages
- Modifying a Message

Logging a Message

The following sample code shows a log message named `Stuff.xml`:

```
<?xml version="1.0"?>
<!DOCTYPE message_catalog SYSTEM "msgcat.dtd">
<message_catalog
  i18n_package=weblogic.stuff
  subsystem=STF
  baseid="500000">
  <logmessage
    severity=warning
    messageid=501234
    method="logNoFile(String name)"
    <messagebody>
      Unable to open file, {0}
    </messagebody>
    <messagedetail>
```

```
      The file, {0}, does not exist. In its absence default
      values will be used and a new version of the file will
      be created automatically.
    </messagedetail>
    <cause>
      The file was deleted.
    </cause>
    <action>
      If this error repeats then investigate unauthorized
      access to the file system. Consider stiffening the
      permissions on the file and directory.
    </action>
  </logmessage>
</message_catalog>
```

The following sample code shows how the `weblogic.stuff` package logs message `1234`:

Code in the `weblogic.stuff` package would log message 1234 as follows:

```
package weblogic.stuff;
import weblogic.stuff.StuffLogger;
import java.io.FileInputStream;
. . .
try {
  FileInputStream fis = new FileInputStream(myfile);
  . . .
}
catch (FileNotFoundException fnfe) {
StuffLogger.logNoFile(myfile);
}
```

Creating a New Catalog

To create a catalog, create a file with the Message Editor, or a text editor. For information about using the Message Editor, see Chapter 4, “Using the BEA WebLogic Server Internationalization Tools and Utilities.”

Because a catalog is an XML file, the format must conform to the syntax specified in the `msgcat.dtd`. A template for log messages is provided in `logmessage_template`. Once created, the catalog must be processed by `i18ngen,` as described in Chapter 4, *Using the BEA WebLogic Server Internationalization Tools and Utilities* section "Creating a New Catalog."

The following points are important to consider when creating a new catalog:

1. The name of the file should follow Java class naming standards.
2. Messages should be added in numerical order.
3. All messages IDs must be larger than the base ID.
4. The base ID is specified not only in the catalog but in the `Msgids` file.

Once created the catalog must be processed by `i18ngen` as described in Chapter 4, *Using the BEA WebLogic Server Internationalization Tools and Utilities* section "Creating a New Catalog."

For easier management, all catalogs should reside in the same source directory.

Creating New Messages

Add the new message definition to the end of the catalog using a message ID one number greater than the last message defined. Select a `method` name that is unique within the catalog. Make sure the new ID is within the range specified for the catalog in the `Msgids` file. If it is not (for example, you have run out of IDs) then you will have to create a new catalog, or extend the range of your catalog using the `endid` attribute.

Once the changes have been made then the catalog must be processed by `i18ngen` as described in "Entering a New Log Message" in Chapter 4.

Modifying a Message

In general, only the message body, message detail, cause and action data should be changed within an existing message. This restriction is necessary because changing values in any other fields may result in code in the field not interfacing correctly (for example, passing the wrong number or type of arguments).

Reprocess the catalog by runniing the `i18ngen` command after modifications to the catalog are complete.

A Localizer Class Reference for BEA WebLogic Server

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

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

Localizers are extensions to the `java.util.ListResourceBundle` class. Four additional methods are provided to simplify the access of the localization data in the Localizer. These methods are:

`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".

The recognized key values correspond to message attributes, and are:

- `Localizer.SEVERITY`
- `Localizer.MESSAGE_ID`
- `Localizer.MESSAGE_BODY`
- `Localizer.MESSAGE_DEATIL`
- `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. Severity values returned are :

- `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 text returned can be further expanded via `java.text.MessageFormat.format()`. This is certainly the case for the message body, detail, cause and action elements, all of which are localizable. The other elements (message ID, severity and subsystem) are not localizable and do not require further processing by MessageFormat.

Lookup Properties

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

```
/weblogic/msgcat/i18n.properties.
```

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

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

The arguments on this line are defined as follows:

- *`nnnnnn`* is the message ID
- *`subsystem`* is the related subsystem
- *`Localizer class`* is the name of the generated `Localizer` class

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

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

Index