# Translating Messages 6

This chapter explains how to create and install message catalogs for your internationalized application. Other aspects of localization, such as formatting and collation, are provided to you along with any SunOS localization packages you have installed. Your translated user messages together with a localization package allow your internationalized application to function properly for a given locale.

## Creating a Message File

Follow these steps to create a message catalog:

1. Use `xgettext`(1) to generate a `.po` (portable message) file. This is a readable text file intended for editing.
2. Translate each `msgid` string in the portable message file, placing the translation on the `msgstr` line directly below it.
3. Run the `msgfmt`(1) program on the portable message file to create a `.mo` (message object) file.

   This is a data file containing a binary message tree for quick machine access. When a program calls `gettext()` or `dgettext()`, the message routines find the appropriate translated string. Your message object file should be called *domain*.`mo`, where *domain* is the name of the text domain you selected.

4. Install the message object file—*domain*.mo—into the LC_MESSAGES directory. This might be under /usr/lib/locale/$*LANG*, but more likely will be the directory specified by your program's bindtextdomain() call.

## The Portable Message File

First run the xgettext command to extract gettext() strings.

```
% xgettext -d demotext demotext.c
```

The output from xgettext is a portable message (suffix .po) file for each domain specified. For the simple "Hello world" example presented in Chapter 4, you would execute xgettext on the source code demotext.c, resulting in an output file called messages.po. In this example, messages is the default domain name. This file would contain:

```
msgid "Hello World!\n"
msgstr

msgid "Goodbye.\n"
msgstr
```

A translator provides the translations by editing this file, appending the target language text to the msgstr lines. If the target language were French, the portable object file might then look like this:

```
msgid "Hello World!\n"
msgstr "Bonjour, tout le monde!\n"

msgid "Goodbye.\n"
msgstr "Au revoir.\n"
```

Lines beginning with a sharp (#) are comments. Empty lines are ignored. Statement lines contain a directive and value. A directive always starts at the beginning of line, and is separated from its value by white space.

If your application program calls `textdomain`() or `dgettext`() to set the message domain, then `xgettext` produces a *textdomain*.mo file for each text domain in the source code.

The following statements are recognized:

`domain` *domainname*
: States that all following target strings until another `domain` directive are contained in *domainname*, which can be any string up to 255 bytes, containing any character in the portable character set, but excluding slash (/). Until the first `domain` directive, all target strings belong to the default domain, `messages`.

`msgid` *message_identifier*
: Specifies the value of the message identifier associated with the following `msgstr` directive. The *message_identifier* string identifies a target string at retrieval time. Usually this is the untranslated string.

`msgstr` *message_string*
: Specifies the target string associated with the *message_identifier* string declared in the immediately preceding `msgid` directive. Every statement containing a `msgstr` directive must be immediately preceded by a `msgid` statement. Usually *message_string* is the translated message.

Any other form of input-line syntax is illegal. Message strings and message identifiers can contain special characters and escape sequences as defined in the following table:

*Table 6-1* Escape Sequences

| Description | Symbol |
|---|---|
| <newline> | \n |
| <tab> | \t |
| <vertical-tab> | \v |
| <backspace> | \b |
| <carriage-return> | \r |
| <form-feed> | \f |
| <backslash> | \\ |

*Table 6-1* Escape Sequences (Continued)

| Description | Symbol |
|---|---|
| <double-quote> | \" |
| octal bit pattern | \ddd |
| hex. bit pattern | \xDD |

The escape sequence *ddd* consists of a backslash followed by 1, 2, or 3 octal digits specifying the value of the desired character. The escape sequence *xDD* is similar but introduces a hexadecimal value. If the character following backslash is not specified in the table above, the effect is undefined.

## *The Message Object File*

Running a file in portable message format through the `msgfmt` utility produces a binary version of the file that can be installed in the locale directory (for example `$APPHOME/locale/$`*LANG*`/LC_MESSAGES`). This file can then be accessed by `gettext()` at run time.

After you have saved the edited `.po` file, run the `msgfmt` command.

```
% msgfmt demotext.po
```

The output from `msgfmt` is a message object file whose name is the domain name, suffixed with `.mo`. The `msgfmt` utility generates one `.mo` file for each text domain in the `.po` file(s). In the example above, executing `msgfmt` on the file `messages.po` produces a binary file name `messages.mo`. Install this file into the appropriate `LC_MESSAGES` subdirectory, renaming it to the proper *textdomain* as necessary.