Internationalization Features 4

This chapter focuses on the general issues of internationalizing applications, and how OLIT address these issues. It also provides a simple example application, which demonstrates how OLIT makes it easy to create and use internationalized applications.

Introduction

As the international market for software becomes increasingly important, software manufacturers need a way to *internationalize* their applications without having to re-engineer or re-compile the application for each language. Ideally, a single version of an application should be able to support any number of languages.

Internationalization refers to the ability of a Graphical User Interface (GUI) to display text in various languages and conventions, and to accept textual input in those languages. *Locale* or *localization* refers to the alphabet and conventions of a particular language or cultural environment. Date, time, and monetary format are examples of locale conventions.

The OLIT toolkit, a user-interface toolkit based on the X Window System and the OPEN LOOK graphical user interface, allows developers to simply and easily create internationalized applications without having to modify the source code for each supported language. An application developed and compiled with OLIT will be able to operate in any of the supported languages and process data according to the rules of that language.

4

System Requirements

OLIT features that support internationalized applications are referred to as *international OLIT* in this chapter. This internationalized version of OLIT supports both European languages (French, German, Italian, and Swedish) and Asian languages (Japanese, Korean, Taiwanese Chinese, and PRC Chinese).

Adding new languages to an international OLIT application consists primarily of setting the default text format and changing messages, labels, and other strings in resource files. For Asian locales, one must also provide localized *input methods* (IM) to allow for multibyte character composition. The input method is the method by which text is entered into the system. Input methods are specific to each Asian language and are provided by Sun Microsystems. They can, however, be redefined and changed by OEMs or independent software vendors.

System Requirements

The first release of international OLIT enables developers to localize to several European and Asian languages. To run this release of international OLIT, you need Solaris®, OpenWindows™, and the *Feature Package* for the locale in which you intend to run it. For example, you need the Japanese Feature Package (JFP) for Japan, the Korean localization package for Korea, and the French localization package for France. These packages consist of extensions to SunOS and incorporate numerous facilities for handling local linguistic and cultural conventions.

Issues Involved in Internationalizing Applications

To internationalize your application, you must address the following issues.

1. ***Locale Setting***. Locale Setting is the method by which the language or cultural environment is set. See “Locale Setting” on page 73.

2. ***Character Encoding***. Conventional applications use 7-bit ASCII encoding to represent each character. However, some languages have larger character sets that require more than the 128 character range permitted by 7-bit encoding; for example, European locales use an 8-bit encoding and Asian locales use extended encoding mechanisms. Character encoding is the method by which a language’s character set is represented. See “Character Encoding and Text Formats” on page 74 and the Sun *Software Internationalization Guide* for further details on character encoding.

3. ***Font Set Handling***. Font handling is simple for the U.S. and European languages that OLIT supports because they only use one character set. However, some languages use multiple character sets and therefore require multiple fonts or a *font set*. See "Font Set Handling" on page 77.

4. ***Localized Text Handling***. The developer needs to be able to use application strings (i.e., error messages, menu text, button text, etc.) in the native language, and have those strings retrieved in the language specified by the locale. See "Localized Text Handling" on page 79.

5. ***Input Method (Asian Locales Only).*** This is the method by which users enter the text of a language. To enter data into a conventional application, the user simply types in the information to be processed. Some languages, however, consist of multiple alphabets which require several keystrokes to create one character. This special handling is called the input method. See "Input Method" on page 80.

6. ***Standards***. Software internationalization is supported by a number of standards organizations. These include IEEE (POSIX), ANSI, X/Open, and the MIT X Consortium. In order to make applications portable across a wide variety of hardware platforms, it is important to use a toolkit that follows these standards as much possible, as international OLIT does. See "Standards" on page 89.

This chapter describes each of these issues in detail and discusses how these issues are addressed by international OLIT.

Locale Setting

The X Toolkit Intrinsics, on which OLIT is layered, provides an application resource called `XnlLanguage` (class `XnlLanguage`) that announces the user's locale to the toolkit and the operating system.

Resource	Type	Default	Access
XnlLanguage	XtNstring	NULL	I

The currently supported values for this resource are: `de`, `fr`, `it`, `ja`, `ko`, `sv`, `zh`, and `zh_TW`. There are three ways to establish the locale, as shown in the following list:

1. Specify `-xnllanguage` on the command line of an OLIT application. For example, to set the locale for an application called *myapplication* to Korean, you type:

   ```
   % myapplication -xnllanguage ko
   ```

2. Specify `*xnlLanguage:` *language* in a X11 Resource Manager database file. For example, to set the locale to traditional Chinese, you add the following line to your `.Xdefaults` file:

   ```
   *xnlLanguage: zh_TW
   ```

3. Set the LANG environment variable in the shell that you are starting the application from. For example, to set the locale to Japanese, you type the following:

   ```
   % setenv LANG ja
   ```

   To set the locale to French, you type the following:

   ```
   % setenv LANG fr
   ```

Establishing the OS locale is the responsibility of OLIT and Xt. You should not use the OS function `setlocale`(3) in your OLIT applications, since OLIT already calls this function internally.

Character Encoding and Text Formats

OLIT supports three character encoding types, or *text formats:*

- Single byte, used in the USA and Europe, which represents each character with one byte
- Multibyte, used in Asian, which represents each character with a variable number of bytes
- Wide character, used in Asian, which represents each character with a fixed number of bytes

Applications can create single-byte, multibyte, or wide character OLIT objects, or a combination of these. If you are writing an application intended for single-byte locales only, you may want to use single-byte text format to avoid the performance overhead incurred by processing multibyte text. However, note that if you use single-byte format, it will be harder to internationalize your application to Asian locales at a later date.

The multibyte text format is fully compatible with ASCII. Since each different character can potentially be a different size, programming with multibyte text can be difficult. However, the multibyte format uses memory efficiently.

Wide character text format, on the other hand, is easier to program since all characters are represented with the same number of bytes. However, the wide character format consumes more storage since it represents all characters with a fixed number of bytes. If all characters are ASCII, many of the bytes will be superfluous.

When deciding which text format is appropriate for a user interface, you should take the following considerations into account:

- Conversion between formats reduces performance.
- OLIT honors the requested text format inside the object implementation.
- Processing multibyte data is inherently more time consuming than processing single-byte or wide character data. Objects that perform intensive data manipulation (for example text-editing object such as OLIT's TextEdit or TextField widgets) will perform better if created as wide character objects. (If an object is certain to only handle 8-bit data, the optimal solution is to create single-byte objects.)

You should design your application so that conversion between formats is minimized. For example, you may want to decide on an object-by-object basis whether textual data will be processed intensively and use wide character format if it is.

Note – The widget makes assumptions based on the text format. For example, if you specify the single-byte format and supply a wide character label, it causes an error.

OLIT widget resources for presentation text are of type `OlStr`. For more information on this type, see "Setting the Default Text Format for an Entire Application" on page 76. To provide support for multiple text formats, international OLIT introduces the `XtNtextFormat` resource. This resource allows you to inform widgets what text format to expect for resources associated with presentation text. This is an interface for programmers only; there is no equivalent interface for users to specify widget text formats. The text format of an object is persistent for the lifetime of the object; it is established at object creation time and cannot be changed.

There are several ways to set the text format:

- Do nothing. If you do nothing, the text format defaults to single byte.
- Set the default text format for the entire application with the `OlSetDefaultTextFormat()` function.
- Set the default text format for an individual widget by changing the widget's `XtNtextFormat` resource.

Setting the Default Text Format for an Entire Application

As a convenience, OLIT maintains a default text format, which widgets inherit when they are created without their text format explicitly specified in the argument list passed to one of the `XtCreateWidget()` family of functions.

For compatibility with previous OLIT releases, the default text format is single-byte unless you change it. To change it, use the `OlSetDefaultTextFormat()` function, which is defined as follows:

```
void OlSetDefaultTextFormat(OlStrRep format);
```

where *format* specifies the character representation of the text. The *format* argument can have the following values:

format Value	Meaning
OL_SB_STR_REP	Single-byte character representation
OL_WC_STR_REP	Wide character representation
OL_MB_STR_REP	Multibyte character representation

Your application should call `OlSetDefaultTextFormat()` immediately after `OlToolkitInitialize()` and before creating the widget hierarchy. Objects subsequently created will have the text format specified in the most recent call to `OlSetDefaultTextFormat()`, unless overridden by explicit arguments. Note that if the application is single-byte only, it does not need to call `OlSetDefaultTextFormat()`.

You can create a widget hierarchy consisting of a combination of multibyte, wide character, and single byte objects. You can do this either by changing the default text format between widget creation calls or by specifying an object's text format explicitly when creating it.

Setting the Text Format for an Individual Widget

OLIT provides a resource, `XtNtextFormat`, which allows you to set the text format for individual widgets. See "XtNtextFormat" on page 29.

Font Set Handling

To represent data that consists of multiple character sets, Release 5 of the X Window System provides the notion of a *Font Set*. An `XFontSet` is a X11R5 data structure that supports this notion. From the user's perspective, an `XFontSet` represents a list of X11 Logical Font Description (XLFD) fonts that allows the application to fully represent the characters used in a particular locale. For full details on Font Sets, refer to the X11R5 documentation.

To specify a font set in a resource file, use a comma-separated list of fonts. For example, you can enter the following in your `.Xdefaults` file:

```
*font: -misc-fixed-medium-r-normal--20-200-75-75-c-100-iso8859-1, \
-jis-fixed-medium-r-normal--16-150-75-75-c-160-jisx0208.1983-0,\
-misc-fixed-medium-r-normal--0-0-75-75-c-0-jisx0201.1976-0
```

When you internationalize your application, you should specify fonts in a resource file. If you specify them within your application, it will be impossible for others to localize it.

OlFont

OLIT supports both the font and font set notions by introducing the `OlFont` type for objects that display text. `OlFont` is an opaque pointer type whose interpretation depends on the setting of its associated `XtNtextFormat` resource. If you create an object as multibyte or wide character, the value of the `OlFont` field will be a valid `XFontSet` identifier. If you create an object as single byte, `OlFont` field will be a valid pointer to an `XFontStruct`.

If the text format is OL_SB_STR_REP and a font set has been specified (using a comma-separated list), the first font in the list will be used to construct an `Xfont` struct.

Setting the Default Font or Font Set

The OLIT widget set provides an `XtNolDefaultFont` resource that specifies the default font or fontset for an application. This resource is discussed in "XtNolDefaultFont" on page 14.

If no font is specified for a widget's `XtNfont` resource, `XtNolDefaultFont` determines the widget's font. In international OLIT, the default value of `XtNolDefaultFont` is determined as follows:

- In the C locale and European locales, the default value of `XtNolDefaultFont` is Lucida sans serif with `Resolution_X` and `Resolution_Y` set to 75.
- In Asian locales, the default value of `XtNolDefaultFont` is the font set required to display all the characters in the code set of the locale.
- An internationalized OLIT application running in Asian locales must define the `XtNolDefaultFont` resource in its locale-specific `app-defaults` file. If the `XtNolDefaultFont` resource is not specified with a list of fully-defined font names the application's startup time will be increased significantly. The application's startup time is dependent on the number of wildcards in the XLFD font names. Use completely specified XLFD font names to optimize performance at startup time. The following is an example of how to set the `XtNolDefaultFont` resource in the Japanese locale:

```
*olDefaultFont: \
-sun-gothic-medium-r-normal--16-140-75-75-c-70-jisx0201.1976-0, \
-sun-gothic-medium-r-normal--16-140-75-75-c-140-jisx0208.1983-0
```

Getting the Default Font or Font Set

The `OlGetDefaultFont()` convenience routine enables you to get the font or font set that will be used if you do not set the `XtNfont` resource for a widget. The syntax of `OlGetDefaultFont()` is:

```
OlFont OlGetDefaultFont(widget w);
```

where *w* is a widget in the application for which you want to get the default font.

If `XtNolDefaultFont` specifies a font that is not available, `OlGetDefaultFont()` returns a null pointer.

Localized Text Handling

OLIT provides resources that enable you to set various text strings (messages, menu labels, button labels, etc.) in an application. For information on the text string resources for a particular widget, see the reference section for the widget. When you create an internationalized application, you should remove all these resources from your application and keep them in locale-specific resource files.

Using Internationalized Help

You can register multibyte help text for a widget with OLIT's `OlRegisterHelp()` function. To register single-byte help, use multibyte help. The OLIT Help API currently supports multibyte only. A wide character API will be added in the future. The syntax for `OlRegisterHelp()` is as follows:

```
void OlRegisterHelp(
    OlDefine    id_type,
    XtPointer   id,
    String      tag,
    OlDefine    source_type,
    XtPointer   source);
```

To use `OlRegisterHelp()` with international OLIT, specify one of the following values for the *id_type* argument.

id_type **Value**	**Meaning**
OL_WIDGET_HELP	Specifies multibyte format help text for an individual widget (for backward compatibility).
OL_FLAT_HELP	Specifies multibyte format help text for a flat widget (for backward compatibility

If you specify OL_DISK_SOURCE for the *source_type* argument, you must specify a single-byte or multibyte filename for the *source* argument.

Help searches the directories specified by XFILESEARCHPATH for the specified filename. If you set the filename from within your program as an absolute path, help ignores XFILESEARCHPATH. If the XFILESEARCHPATH expansion does not find a file for help, the current directory is searched. Within the value

of XFILESEARCHPATH, any instance of the string "%T" is expanded to "help" and "%N" is expanded to the filename the programmer specifies as the source parameter to `OlRegisterHelp()`.

For a description of the other arguments of `OlRegisterHelp()`, see "Help Function" on page 146.

Localized Messages

OLIT issues textual error message in the program's locale at startup. To do this, OLIT registers a private language procedure with the Intrinsics during `OlToolkitInitialize()`; see page 92.

Do not register an application-specific language procedure. If you do, it will interfere with OLIT's locale-announcement mechanism and you will have to take responsibility for it in your application. If your application requires processing before OLIT's language procedure is called, you can provide your own procedure and call OLIT's procedure from it. To get the language procedure registered by OLIT, call:

```
olit_proc = XtSetLanguageProc(NULL, NULL, NULL);
```

Multibyte and Wide Character Text Buffer Functions

International OLIT provides multibyte and wide character equivalents to the single-byte text buffer functions in previous releases. A list of these functions and their syntax are provided in "Text Buffer Functions for Internationalization" on page 176.

Input Method

The input method (IM) is the algorithm by which users enter the text of a language. The input method for each language may be different, depending on the linguistic structure and conventions of that language.

International OLIT follows the *X Window System Version 11, Input Method Specification, Draft 3.0*. This specification was derived as a result of discussions among X Consortium members on standardizing the input handling of characters in various languages by X clients.

Input Method

For many languages, there isn't a one-to-one key to character mapping, regardless of how the keyboard is configured. In order to support such languages, an input method is required.

In English and European languages, users enter the desired text by typing in a sequence of letters to create a word. However, for Asian languages based on ideographic characters, input is more complicated. For example, there are two phonetic alphabets in Japanese—Hiragana and Katakana—and the traditional ideogrammatic alphabet, Kanji. In any piece of writing, all three alphabets may be used. Japanese words can also be spelled out phonetically in English. This is called Romaji.

To handle European language characters where there is no key on the standard keyboard that maps to the desired character, use the Compose key to initiate a composed character sequence (e.g., Compose key, <letter>, <accent or diacritical mark>).

To handle languages for which there isn't a one-to-one key to character mapping, input methods provide features such as the following:

- A control key sequence, which selects the *input mode*
- A pre-edit region, which displays characters as the user enters them but before the user *commits* them
- A lookup choice region, which displays ideographic characters and allows the user to choose one
- A status region, which provides information such as whether conversion is activated and the state or mode of the input method

Text input widgets, in conjunction with some input methods, also can provide advanced, language-specific, pre-editing features. For instance, the OLIT TextEdit widget can detect certain conditions under which it will commit any uncommitted pre-edit text without the user having to take further action. This technique is known as *implicit commit*.

Example: In a mail application the user enters a message in Japanese and presses a "send" button to dispatch the composed message. If there is uncommitted pre-edit text, the user's intention is that it be part of the message. The pre-edit text has not, however, been committed to the text buffer. In this case it is useful for the toolkit to intervene and cause a commit to occur before the application processes the buffer to send the message.

Details of which operations trigger implicit commit semantics in OLIT widgets can be found in the localization documentation for the appropriate languages.

The use of these features varies with the input method. For more information, see the documentation for the input method you are using.

Figure 4-1 shows the input method screen regions.

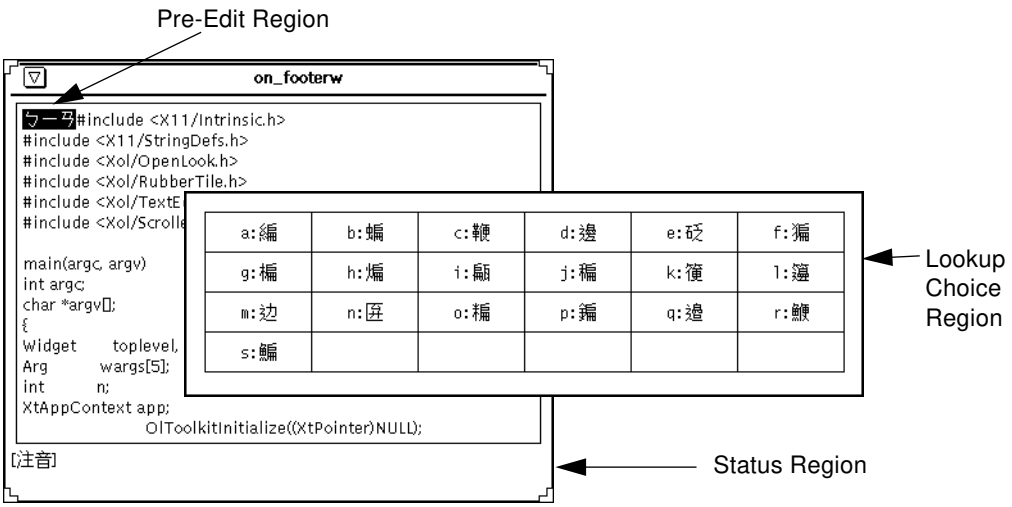

Figure 4-1 Input Method Screen Regions for `zh_TW` locale

Setting the Input Method Pre-Edit and Status Styles (Asian Locales Only)

There are two aspects of the input method that an OLIT application can control:

- The *pre-edit style*, which specifies where and how pre-edit data is presented. The pre-edit style can vary from widget to widget within a shell.
- The *status style*, which specifies where status feedback is presented. The status style, unlike the pre-edit, is an attribute of the shell and is expected to remain the same across all widgets inside the shell.

OLIT provides two new resources that specify the pre-edit and status styles: `XtNimPreeditStyle` and `XtNimStatusStyle`, described in the following sections.

XtNimPreeditStyle (Asian Locales Only)

The `XtNimPreeditStyle` resource selects the pre-edit style. This resource is supported by all the OLIT widgets that allow text input. See, for example, TextEdit ("XtNimPreeditStyle" on page 635).

Resource	Type	Default	Access
XtNimPreeditStyle	OlImPreeditStyle	OL_NO_PREEDIT	GI

If the specified style is not supported by the input method, the ability to pre-edit is lost. The currently supported pre-edit styles are:

XtNimPreeditStyle Value	Meaning
`OL_ON_THE_SPOT/ "onTheSpot"`	IM directs the application to display the pre-edit data
`OL_OVER_THE_SPOT/ "overTheSpot"`	IM displays pre-edit data in its own window
`OL_ROOT_WINDOW/ "rootWindow"`	IM displays pre-edit data outside the application in a window that is a child of the base window
`OL_NO_PREEDIT/"none"`	IM does not display pre-edit data

The following figure shows an example of the `onTheSpot` pre-edit style. The pre-edit data is shown in reverse video. When the user commits the data, it is sent to the client and displayed in normal video.

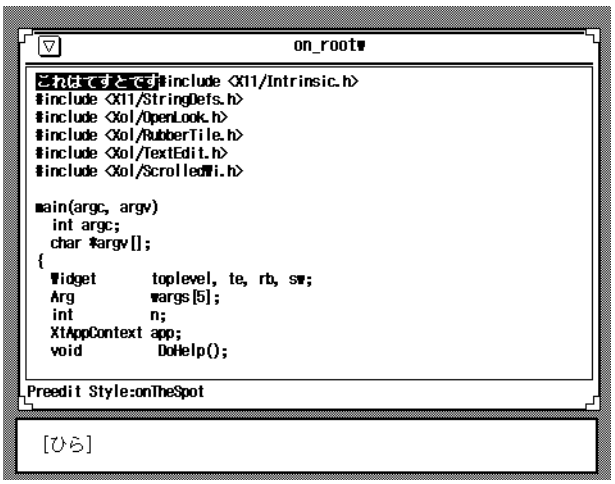

Figure 4-2 `onTheSpot` Pre-Edit Style for `ja` locale

The following figure shows an example of the `overTheSpot` pre-edit style.

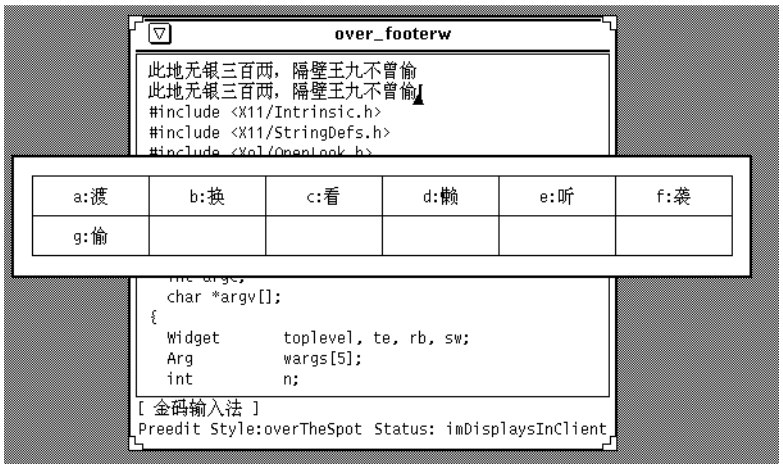

Figure 4-3 `overTheSpot` Pre-Edit Style for `zh` locale

The following figure shows an example of the `rootWindow` pre-edit style.

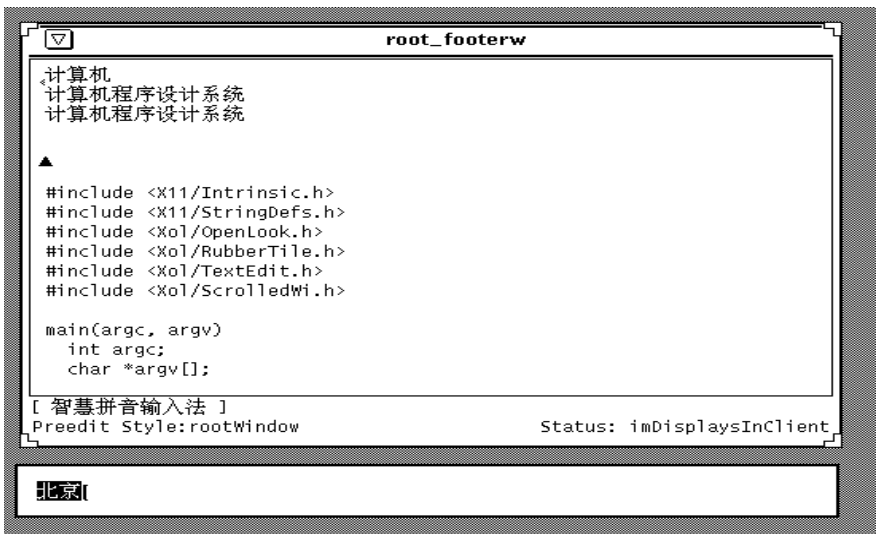

Figure 4-4 `rootWindow` Pre-Edit Style for `zh` locale

See the X11R5 documentation for a full description of each of the pre-edit styles.

XtNimStatusStyle (Asian Locales Only)

The `XtNimStatusStyle` resource, defined in the `VendorShell` class, determines the style of IM status feedback.

Resource	Type	Default	Access
XtNimStatusStyle	OlImStatusStyle	OL_NO_STATUS	GI

The supported styles are:

XtNimStatusStyle Value	Meaning
`OL_IM_DISPLAYS_IN_CLIENT/ "imDisplaysInClient"`	The IM generates status feedback in the footer of the shell window
`OL_IM_DISPLAYS_IN_ROOT/ "imDisplaysInRoot"`	The IM generates status feedback in a separate window
`OL_NO_STATUS/"none"`	The IM doesn't generates any status feedback

When `XtNimStatusStyle` is set to OL_IM_DISPLAYS_IN_CLIENT, a number of other resources are available to set characteristics of the application footer. See "Application Resources for the IM Footer (Asian Locales Only)" on page 87.

The following figure shows an example of the `imDisplaysInClient` status style.

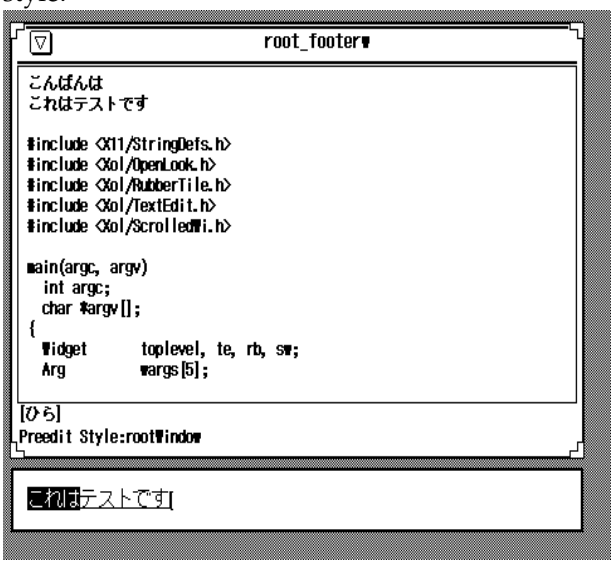

Figure 4-5 `imDisplaysInClient` Status Style for `ja` locale

The following figure shows an example of the `imDisplaysInRoot` status style.

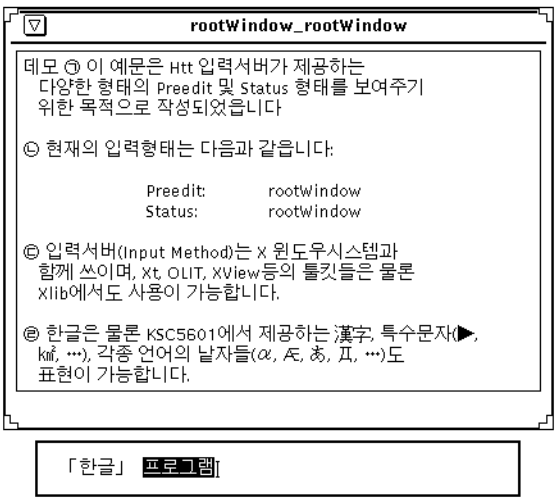

Figure 4-6 `imDisplaysInRoot` Status Style for `ko` locale

Application Resources for the IM Footer (Asian Locales Only)

Resource	Type	Default	Access
XtNimFontSet	OlFont	XtDefaultFont	SGI
XtNdefaultImName	String	NULL	SGI

XtNimFontSet (Asian Locales Only)

The `XtNimFontSet` resource specifies the internationalization IM status footer's font set.

XtNdefaultImName

The `XtNdefaultImName` resource specifies the string to identify the IM Server.

XtNshellTitle

`XtNshellTitle` is an `OlStr` resource that allows the title of shell widgets to be set. The shell being set must be a subclass of `VendorShell`. The usage of this resource is analogous to the `XtNtitle` resource defined by the Intrinsic classes. `XtNshellTitle` and `XtNtitle` are synchronized by OLIT; calling `XtSetValues()` on either will cause both to be updated.

Example

To internationalize a simple OLIT application for the European or Asian locales, you need to:

1. **Remove any resources that contain display text from the application code and put them in a resource file.**
2. **Specify the text format for widgets that display text (by specifying a default text format for the application or by specifying the text format for individual widgets).**
3. **Specify a font set for resources that contain `OlFont` values (such as `XtNfont`) in the resource file.**

4

Input Method

Suppose you want to write an internationalized application that displays a StaticText widget with some wide character text in it. To do this, you would write code similar to the following.

```
#include <X11/Intrinsic.h>
#include <X11/StringDefs.h>
#include <Xol/OpenLook.h>
#include <Xol/StaticText.h>

main(argc, argv)
    int     argc;
    char    *argv[];
{
    Widget toplevel, msg_widget;
    XtAppContext app;

    /* Initialize the OLIT toolkit */
    OlToolkitInitialize((XtPointer)NULL);

    /* Set the default text format to wide character */
    OlSetDefaultTextFormat(OL_WC_STR_REP);

    toplevel = XtAppInitialize(&app, "Memo",
                               (XrmOptionDescList)NULL,
                               0, &argc, argv, NULL,
                               (ArgList) NULL, 0);

    /* Create a staticText widget. */
    msg_widget = XtVaCreateManagedWidget("msg",
                                  staticTextWidgetClass,
                                  toplevel,
                                  NULL);

    /* Realize the widgets and enter the event loop. */
    XtRealizeWidget(toplevel);
    XtAppMainLoop(app);
}
```

Note that this application does not specify the string that appears in the StaticText widget. You specify this text in the resource file as follows:

```
*StaticText.string: <text>
```

Also note that this application is different from a conventional OLIT application in that it calls `OlSetDefaultTextFormat()` to set the default text format. `OlSetDefaultTextFormat()` sets the default text format for any widgets in the application that display text.

If you only want to set the text format for an individual widget, you set its `XtNtextFormat` resource. For example, in the application above you would create the StaticText widget as follows:

```
msg_widget = XtVaCreateManagedWidget("msg",
                                     staticTextWidgetClass,
                                     toplevel,
                                     XtNtextFormat, OL_WC_STR_REP,
                                     NULL);
```

For some locales, you may also need to specify a font set for the application. To do this, you add the following line to the resources file:

```
*StaticText.font:  font set
```

The following figures show the application with Korean and Japanese text.

Figure 4-7 "Hello World" in Korean

Figure 4-8 "Hello World" in Japanese

Standards

There are three standards that international OLIT follows:

- Internationalized UNIX
- MIT X11 Internationalization Standard
- Internationalized Extension of OPEN LOOK Specification

4

Standards