Input Method 5

An input method is a method by which an application directs the user to type, select, and send text to an application. Input methods differ for each language depending on the language's structure and conventions. Input methods for Japanese, Chinese, and Korean are provided by SunSoft.

The current XView programming environment follows the X Window System Version 11 Release 5 specifications for input methods. Refer to *Xlib - C Language X Interface MIT X Consortium Standard X Version 11, Release 5* for additional information. XView supports the X input method in panels, tty subwindows, text subwindows, and canvases.

Purpose of Input Methods

English text is entered into an application directly by typing letters from the keyboard. To enter text in the Asian locales, an input method is required because users cannot enter all characters into an application directly from the keyboard. It is often impractical to map all Asian alphabets and characters on to a keyboard; many Asian languages have extremely large character sets and several alphabets.

- Japanese text uses three different writing systems: Hiragana, Katakana, and Kanji. Hiragana and Katakana are phonetic alphabets. Users enter them directly from the keyboard using Romaji, a way to spell out Hiragana and Katakana with a western alphabet. Hiragana and Katakana combinations can be converted to ideographic Kanji characters.

- Korean uses two different writing systems: Hangul and Hanja. Hangul is a phonetic alphabet users can enter from the keyboard and then convert to Hanja (ideographic) characters.
- Chinese employs numerous input methods including phonetic spelling, stroke combinations, and phrase compositions.

For further information on specific input method operations, refer to the *JFP User's Guide*, or the Asian Solaris 2.4 user's guide, which are listed in the Preface.

Input Method Operation

In typical Asian language input method(s), the following occurs:

1. The user selects a phonetic alphabet in which to enter characters.
2. The user types the word, which appears in inverse video in an area of the screen called the *preedit region*.
3. To convert the word in the preedit region to another alphabet or an ideographic character, the user presses the Select Start key.
4. Phonetically equivalent choices are displayed in the *lookup choice region*, and the user selects the most appropriate choice to replace the word in the preedit region.

Input Method Screen Regions

Each Asian language has its own input method, but the screen regions are similar from language to language. Japanese, for example, has three screen regions:

- *Preedit region*
  The preedit region is activated when input method conversion is enabled. Entered text is displayed in inverse video. When preedit text is committed, the text is sent to the client and displayed in normal video.
- *Lookup choice region*
  In many Asian languages one phonetic representation of a word can have several ideographic representations. The lookup choice region displays the multiple ideographic choices that correspond to one phonetic

representation. For example, in Japanese, the user can type in a word phonetically, then display the lookup choice region, and finally, select the appropriate Kanji, Hiragana, or Katakana representation.

- *Status region*
  The status region provides feedback on the state of the input method. Some languages are very complex and have several input methods. For example, in Chinese, users can choose from TsangChieh, Chuyin, ChienI, Neima, ChuanHsing, or Telecode input methods. In the Japanese input method, the status region displays alphabet (Hiragana or Katakana) that is being used. The status region is part of the frame window and is displayed above the frame footer.

The screen regions for the Japanese input method for the current XView release are shown in Figure 5-1.

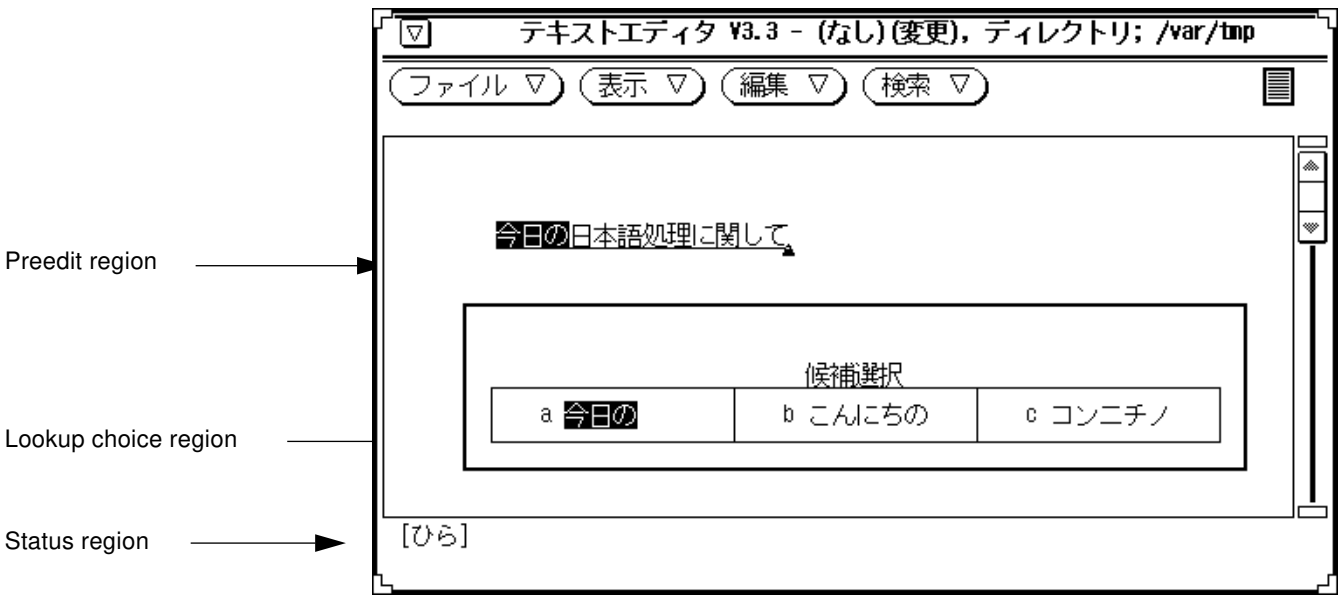

Figure 5-1 Japanese Input Method Screen Regions

5

Input Method Styles

Xlib supports a variety of input method styles, which allow for different user interaction and display models for preedit and status regions. XView supports many of these styles. In particular, the current XView release supports the following `XIMStyle` values:

- Xlib preedit styles:
  - XIMPreeditCallbacks
  - XIMPreeditPosition
  - XIMPreeditNothing
  - XIMPreeditNone
- Xlib status styles:
  - XIMStatusCallbacks
  - XIMStatusArea
  - XIMStatusNothing
  - XIMStatusNone

Specifying Styles

Input method styles can be specified in the following ways (listed in order of precedence):

- XView attribute
- User-specified command line options
- User-specified locale-specific X resources (`~/.Xdefaults`)
- User-specified X resources (`~/.Xdefaults`)

XView attributes override user-specified styles, and command line entries override X resource settings. Table 5-1 on page 35 and Table 5-2 on page 35 show the preedit and status style values. By default, XView requests the use of an on-the-spot preedit style and a client-displays status style.

Table 5-1 Preedit Style Values

XView Attribute	Command Line Option	X Resources
WIN_X_IM_STYLE_MASK	-preedit_style	OpenWindows.ImPreeditStyle
XIMPreeditCallbacks	onTheSpot	onTheSpot
XIMPreeditPosition	overTheSpot	overTheSpot
XIMPreeditNothing	rootWindow	rootWindow
XIMPreeditNone	none	none

Table 5-2 Status Style Values

XView Attribute	Command Line Option	X Resources
WIN_X_IM_STYLE_MASK	-status_style	OpenWindows.ImStatusStyle
XIMStatusCallbacks	clientDisplays	clientDisplays
XIMStatusArea	imDisplaysInClient	imDisplaysInClient
XIMStatusNothing	imDisplaysInRoot	imDisplaysInRoot
XIMStatusNone	none	none

Determining the Default Style

XView clients can request a particular input method (IM) style; however, the requested style is only considered to be a hint. The actual IM styles used by the application depend on what styles are supported by both the toolkit and the input method server.

XView attempts to accommodate the requested IM style. If, however, the style requested is not supported, the default IM style is set to a root-window preedit style and an im-displays-in-root status style.

The attribute XV_IM_STYLES can be used to determine what styles are supported. It returns an XIMStyles structure.

For further details on specifying IM styles, see “Windows: Handling Input” on page 71.

5

Enabling and Disabling the Input Method

If you expect a window to use Asian text input, request the use of an input method during `xv_create()`. The `WIN_USE_IM` attribute is considered to be a hint for enabling or disabling the input method for a given window. If an input method is available, that is, if the locale-specific resource `xview.needIM` is `TRUE`, setting `WIN_USE_IM` to `TRUE` will enable the input method. Setting `WIN_USE_IM` to `FALSE` will disable use of the input method. Refer to Chapter 6, "XView API for Internationalization," in the "Resources" section for more information on `xview.needIM`.

`WIN_USE_IM` can be set on any frame, panel, tty subwindow, text subwindow, or canvas. By default, `WIN_USE_IM` is `TRUE` if the input language specified by `XV_LC_INPUT_LANG` supports an input method. `WIN_USE_IM` is an inheritable attribute; therefore, subwindows inherit the value of `WIN_USE_IM` from the parent frame if it is not set explicitly.

If a subwindow does not require Asian text input—say a panel containing buttons or a read-only text subwindow, create it with `WIN_USE_IM` set to `FALSE`. This avoids having the toolkit create and maintain unnecessary input context (IC) resources and avoids the overhead of connecting with the input method server.

Once an input method is enabled, the user can compose Asian text by interacting with the various input method screen regions.

Input Method Architecture

Internationalized applications receive user text input by communicating with an input method. XView makes a single input method connection with Xlib upon calling `xv_init()` and operates in the specified input language locale.

Different Xlib implementations provide input method support in various ways.[1] Shown in Figure 5-2 on page 37 is one possible example of an application connecting with an Asian input method, which is input method server based. In this example, the client is connected with the input method server using a back-end method. One input method server can provide input method service to multiple X clients.

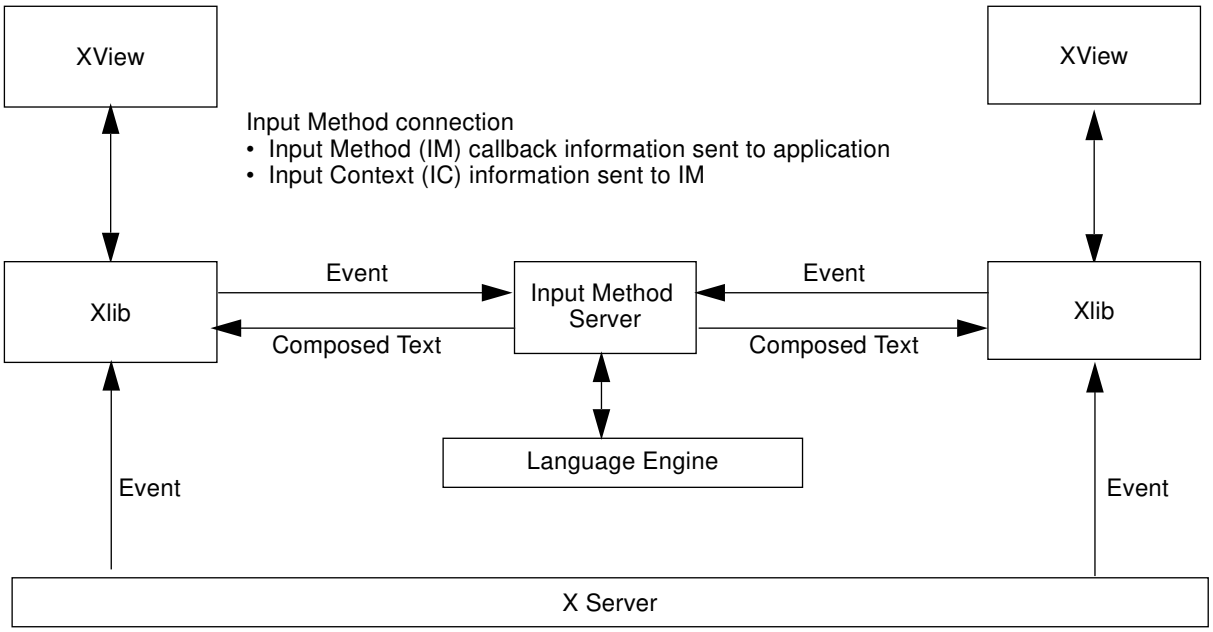

Figure 5-2 High-level Overview of Input Method

By default, XView provides an on-the-spot and client-displays input method style, in which the input method makes requests to the application to display preedit and status information through a series of callback functions.[2] XView

1. Various Xlib implementations exist today to support input methods. For instance, European input methods may not require preedit feedback or dictionary lookup, and may be implemented directly within Xlib. Conversely, many Asian input methods are implemented with Xlib establishing a connection with another process called the input method server. Input method servers may also connect with a language engine process that aids in dictionary lookup. Additionally, an input method can be further characterized as a *front-end* or *back-end method* depending on whether the event is intercepted before it reaches the application or not.

automatically handles communication between the application and the input method by creating an X input context (IC)[1] and registering default preedit callbacks and status callbacks for each panel, canvas, text subwindow, or tty subwindow that has input method enabled.

Only one IC is registered per subwindow, even if there are multiple input areas within a subwindow. For instance, a panel with multiple text items will have each text item share the same IC.

Implicit Commit of Preedit Text

Certain mouse and keyboard actions automatically commit a preedit string without requiring the user to enter a commit key sequence. Implicit commit actions are listed in Table 3-1. In some cases key actions are consumed by the language conversion engine. The language conversion engine can also implicitly commit text. For example, in the Japanese input method, when a preedit string has been converted using the Control-N key, subsequent preedit input implicitly commits the preedit string. Refer to the specific Asian Language Environment documentation (see "Further Documentation" on page xvii) for implicit commit behavior of a particular language's conversion engine.

Implicit commit can also be triggered programmatically by certain attributes and functions in panels and text subwindows. Refer to "Panels" on page 54 and "Text Subwindows" on page 63 for details.

Table 5-3 Implicit Commit Actions

Function Keys	**Panel**	**Textsw**	**Ttysw**
Left arrow key	Y	Y	Y
Down arrow key	Y	Y	Y
Right arrow key	Y	Y	Y

2. In Xlib terminology, the `XIMStyle` of `XIMPreeditCallbacks` and `XIMStatusCallbacks` are supported.

1. In Xlib, there is the concept of the input context (IC), which is essentially an abstraction of a data structure that contains information about the state of an input method area.

Up arrow key	Y	Y	Y
Home key	N	Y	N
End key	N	Y	N
PgUp key	N	Y	N
PgDn key	N	Y	N
Paste	Y	Y	Y
Find	Y	Y	N
Again	N	Y	N
Undo	Y	Y	N
Carriage Return	Y*	Y*	Y*
Tab	Y*	Y*	Y*
Select (left mouse button)	Y	Y	Y
Adjust	Y	Y	N

* Denotes key actions known to be consumed by some language conversion engine

The table above applies to both SPARC and x86 hardware. x86 keyboards have all function keys listed above, except for Paste, Find, Again, and Undo. The x86 functionality can be obtained by:

- Paste—Meta-v
- Find—Meta-f
- Again—Meta-a
- Undo—Meta-u
- Adjust—middle mouse button on three-button mice, and Shift+mouse right button on two-button mice

Customizing Input Method Callbacks

A default user interface is provided for the preedit and status regions when on-the-spot and client-displays styles are used. If you want to customize the user interface, you can specify your own callback functions for preedit and status regions using the `WIN_IC_PREEDIT_*` and `WIN_IC_STATUS_*` attributes. The lookup choice region is not customizable since it is displayed by the input method and not by the toolkit. Refer to “Windows: Handling Input” on page 71 for details on enabling input methods and attributes available for customizing the input method interface.

.

5