XView API for Internationalization 6

This chapter describes internationalization features available within each XView package. You will find detailed discussions about many of the attributes and functions that will help you internationalize your application.

The sections in this chapter, presented alphabetically by packages, correspond with chapters in the O'Reilly *XView Programming Manual*. Use the information in this chapter as an addendum to the O'Reilly manual.

The *XView Programming Manual* discusses the multibyte API, and for the most part, the wide character API behaves similarly. Differences between the two types of attributes and functions are noted here. This chapter also describes additional attributes and functions that specifically aid in supporting internationalization.

Attributes and functions are listed in tables at the beginning of each section for quick reference, and if necessary, they are discussed in further detail.

Canvases

The following sections describe the canvas input method, input context, and the `CANVAS_IM_PREEDIT_FRAME` attribute.

Canvas Input Context

When canvases are created with `WIN_USE_IM` set to `TRUE`, their input method is enabled for all the canvas paint windows. All paint windows share the same input context; therefore, if input method is activated in one paint window, preedit is active in all paint windows. It is not possible to change the state of `WIN_USE_IM` in an individual paint window using `WIN_USE_IM` with `CANVAS_PAINTWINDOW_ATTRS`.

Canvas Input Method

When on-the-spot preedit style is used, the canvas package uses a popup window to display preedit text. The popup window appears near the canvas window. The popup window is used because a canvas provides a drawing surface only, and it is not possible to determine how an application using canvas might implement text rendering. Canvases within a frame share the same preedit popup window. The frame status region and the preedit popup window contents always reflect the state of the canvas that has the focus.

`CANVAS_IM_PREEDIT_FRAME`

You can label, display, position, or size the preedit popup window.

Querying `CANVAS_IM_PREEDIT_FRAME` with `xv_get()` returns the frame handle of the preedit popup window associated with a canvas. The attributes that can be set on the preedit popup window are `XV_LABEL`, `XV_SHOW`, `XV_WIDTH`, `XV_HEIGHT`, `XV_X`, and `XV_Y`.

Cursors

`CURSOR_STRING_WCS` supports text drag and drop cursors for wide character strings.

File Chooser

The default file chooser object contains file list, history, and pathname objects. For further information refer to:

- "File Lists" on page 44
- "History" on page 53

- "Pathnames" on page 57

Table 6-1 lists wide character file chooser attributes.

Table 6-1 File Chooser Attributes

Wide Character Attributes

`FILE_CHOOSER_APP_DIR_WCS`
`FILE_CHOOSER_CUSTOMIZE_OPEN_WCS`
`FILE_CHOOSER_DIRECTORY_WCS`
`FILE_CHOOSER_DOC_NAME_WCS`
`FILE_CHOOSER_FILTER_STRING_WCS*`
`FILE_CHOOSER_NOTIFY_FUNC_WCS`
`FILE_CHOOSER_WCHAR_NOTIFY`

* See the discussion following this table.

Caution – Do not use `FILE_CHOOSER_FILTER_STRING_WCS` for multibyte characters in filenames. Results are unpredictable. This limitation will be removed in future releases.

Callbacks registered by the functions `FILE_CHOOSER_CHANGE_DIR_FUNC`, `FILE_CHOOSER_FILTER_FUNC`, and `FILE_CHOOSER_COMPARE_FUNC` normally receive multibyte character strings, but callbacks can receive wide character strings when the function `FILE_CHOOSER_WCHAR_NOTIFY` is turned on.

The `File_chooser_row` structure changes to the `File_chooser_row_wcs` structure as follows.

```
typedef struct {
    wchar_t *file_wcs;
    struct stat *stats;
    File_chooser_opmatched;
    char *xfrm;
} File_chooser_row_wcs;
```

The xfrm field contains the string returned from the function strxfrm(3). The built-in comparison functions are all defined to have parameters of type File_chooser_row. Since the functions use only the xfrm field, they also work for File_chooser_row_wcs structures. If the client calls these functions directly, it should typecast accordingly.

File Lists

Table 6-2 lists the wide character file list attributes.

Table 6-2 File List Attributes

Wide Character Attributes
FILE_LIST_DIRECTORY_WCS
FILE_LIST_DOTDOT_STRING_WCS
FILE_LIST_FILTER_STRING_WCS*
FILE_LIST_WCHAR_NOTIFY*

* See the discussion following this table.

Caution – Do not use FILE_LIST_FILTER_STRINGS_WCS for multibyte characters in filenames. Results are unpredictable. This limitation will be removed in future releases.

Callbacks registered by the functions FILE_LIST_CHANGE_DIR_FUNC, FILE_LIST_FILTER_FUNC, and FILE_LIST_COMPARE_FUNC normally receive multibyte character strings, but callbacks can receive wide character strings when the function FILE_LIST_WCHAR_NOTIFY is turned on.

The File_list_row_wcs structure is used instead of the File_list_row structure. It is defined as follows:

```
typedef struct {
    File_list  file_list;
    Panel_list_row_values_wcsvals;
    struct stat    stats;
    File_list_opmatched;
    char *xfrm;
} File_list_row_wcs;
```

The `xfrm` field contains the string returned from the function `strxfrm(3)`. The built-in comparison functions are all defined to have parameters of type `File_list_row`. Since the functions use only the `xfrm` field, they also work for `File_list_row_wcs` structures. If the client calls these functions directly, it should typecast accordingly.

Fonts

In the X11 R5 window environment, multiple fonts are required to support languages with multiple character sets, usually one font per character set. Therefore, Xlib created the concept of font sets to support multiple fonts; and the font package in XView now handle font sets. A font set object is a collection of one or more fonts needed to display characters in languages with multiple character sets. XView handles all font operations with font sets; however, fonts are also supported for backward compatibility.

The actual set of fonts underlying a particular abstract font set may vary from one locale to another. A font set is defined by the XLFD names of the fonts in the font set. The definition of a font set can be specified in a program or in a font set database using font set names. OpenWindows™ provides some default locale-specific font set database files:

$OPENWINHOME/lib/locale/<locale>/OW_FONT_SETS/OpenWindows.fs

Applications can define dedicated font set databases. Refer to Appendix C, “Font Set Definitions for further details on defining a font set database.

An XView application gets a font set object by calling `xv_create()` or `xv_find()` in a manner analogous to creating or finding a font. All objects created by the font package are font sets, except for glyph fonts. The font set object works with the existing API in the font package. However, some existing attributes can only support C locale fonts and fonts for locales that use the ISO Latin-1 character set. New attributes support the font set object.

Some locales, such as the English (C) or German (de) locales, require only one font. In such cases, a font set of one member is created. Therefore, C locale and locales that use the ISO Latin-1 character set are not required to use the font set definition database. The font package does not use the font set definition database for these locales.

6

Internationalized drawing functions have been created in R5 Xlib to accommodate the concept of font set objects. The font set object does not work within the frame work of the Graphics Context (GC). X programs usually set or change fonts through the `XCreateGC()` or `XChangeGC()` functions. The internationalized Xlib drawing functions, such as `XwcDrawString()`, accept a font set parameter directly.

Font Set API

Table 6-3 lists font set attributes that support wide characters:

Table 6-3 Font Set Attributes

Attributes
FONT_CHAR_WIDTH_WC
FONT_CHAR_HEIGHT_WC
FONT_COLUMN_WIDTH*
FONT_LOCALE*
FONT_NAMES*
FONT_SET_ID*
FONT_SET_SPECIFIER*
FONT_STRING_DIMS_WCS

* Supports font set objects. See the discussion following this table.

A font set contains many fonts needed to render the multiple character sets. Therefore, the attributes `FONT_DEFAULT_CHAR_WIDTH` and `FONT_DEFAULT_CHAR_HEIGHT` mean the combined default width and height for all the fonts in the font set; based on the dimensions obtained from the Xlib function `XExtentsOfFontSet()`.

`FONT_FAMILY`, `FONT_SCALE`, `FONT_SIZE`, `FONT_STYLE`, `FONT_RESCALE_OF`, `FONT_SIZES_FOR_SCALE`, and `FONT_TYPE` are also supported for font set objects.

FONT_COLUMN_WIDTH

`FONT_COLUMN_WIDTH` enables an application to maintain the same screen column display widths of fixed-width fonts across various locales.

Querying the value of FONT_COLUMN_WIDTH returns the width of a column in pixels. The widths of Asian characters vary depending on the character set used. In the C and ISO Latin-1 environments, a character is one byte; with fixed-width fonts, a character occupies one column on the screen. In Asian locales, one character may be more than one byte, and it may occupy many columns on the screen. Refer to "EUC Programming Issues" on page 27 for details.

FONT_LOCALE

This attribute specifies the locale of a font set. The locale information is required to determine the set of fonts to be used in creating the font set object. The default value of FONT_LOCALE is the locale associated with the XV_LC_BASIC_LOCALE attribute. An example of a font set created in the display language locale is shown below:

```
Xv_Font font_set;

font_set = (Xv_opaque) xv_create(NULL,
    FONT_FAMILY, FONT_FAMILY_SANS_SERIF,
    FONT_LOCALE, xv_get(frame, XV_LC_DISPLAY_LANG),
    NULL);
```

FONT_NAMES

A list of font names for constructing the font set object can be specified using this attribute. The font names should be specified in the X11 Logical Font Description (XLFD) format. The list of font names has to be NULL terminated. The fonts of this list should satisfy all the character sets for the specific

FONT_LOCALE. For example, the required fonts for the Korean locale must satisfy the character sets ksc5636 and ksc5601.1987. An example below shows how FONT_NAMES can be used:

```
char *font_names[] = {
    "-sun-gothic-medium-r-normal--14-120-75-75-c-60-ksc5636-0",
    "-sun-gothic-medium-r-normal--14-120-75-75-c-120-ksc5601.1987-0",
    NULL};
Xv_Font font_set;

font_set = xv_create(NULL, FONT,
    FONT_NAMES, font_names,
    NULL);
```

FONT_SET_ID

The internationalized text functions in X11 R5 are necessary for rendering or querying the dimensions of multibyte or wide character strings. The font set parameter required by these internationalized text functions can be obtained from an XView font set object by querying the FONT_SET_ID attribute. The font set of a font set object is analogous to the XID of a font. Detailed information concerning internationalized text functions can be found in the X11, Release 5 documentation for Xlib. The following example shows how to use the value of FONT_SET_ID in a wide character drawing routine:

```
Display     *display;  /* Specifies connection to the X server */
Drawable    xid;    /* Specifies the drawable */
GC          gc;     /* Specifies the GC */
int         x, y;   /* X and Y coordinates */
wchar_t     *string;   /* Wide character string */
int         num_wchars;    /* Number of wide characters */
XFontSet    font_set_id;   /* font set */
Xv_Font     font_set;      /* XView font set object */

font_set_id = (XFontSet) xv_get(font_set, FONT_SET_ID);
XwcDrawString(display, xid, font_set_id, gc, x, y, string, num_wchars);
```

FONT_SET_SPECIFIER

A font set can also be specified through one of the following:

- `FONT_SET_SPECIFIER` attribute
- Command line option `-font`
- X resources: `OpenWindows.MonospaceFont`, `OpenWindows.RegularFont`, and `OpenWindows.BoldFont`

The font set specifier is a shorthand way of specifying a list of fonts for a particular locale. For example:

```
Xv_Fontfont_set;
font_set = xv_find(frame, FONT,
    FONT_SET_SPECIFIER,"gotm14",
    FONT_LOCALE, locale,
    NULL);
```

In this example, the font set definition database will be queried for a font set named `gotm14`. The value associated with the `FONT_SET_SPECIFIER` attribute is processed in the following manner:

1. Use the value as a font set name to find a definition in the font set definition database.
2. If a font set definition corresponding to the font set specifier is not found in the font set definition database, use the value as an XLFD font name. Assuming the locale requires only one font and the XLFD name exists on the X11 server, a font set object will be created containing the specified font.
3. If 1 and 2 fail, `NULL` is returned.

The command line option `-font` takes the name of a font set object, the value of `FONT_SET_SPECIFIER`. It is dissimilar to the value of `FONT_NAME`.

Locale-specific X resources can be used to specify the font set name:

OpenWindows.MonospaceFont.<basiclocale>
OpenWindows.RegularFont.<basiclocale>
OpenWindows.BoldFont.<basiclocale>
font.name.<basiclocale>
Font.Name.<basiclocale>

where <basiclocale> is the basic locale setting. Font.Name and font.name are provided for backward compatibility. OpenWindows.*Font resources take precedence, and therefore, override the values of Font.name. and font.name. See "Resources" on page 58 for detailed information.

Glyph Fonts

Glyph fonts are generally not specific to a locale, so XView does not create font set objects for glyph fonts.

For applications using private glyph fonts or special fonts, such as -itc-zapfdingbats-medium-r-normal--*-140-*-*-p-*--dingbats font, create the font with the attribute FONT_TYPE as below:

```
font = (Xv_Font)xv_find(frame, FONT,
    FONT_NAME, "-itc-zapfdingbats-medium-r-normal--*-140-*-*-p-*--dingbats"
    FONT_TYPE, FONT_TYPE_GLYPH,
    NULL);
```

These font objects created with FONT_TYPE_GLYPH are not font set objects. Applications should obtain XV_XID of the font objects and use font related functions from Xlib, such as XDrawString(); and not font set related Xlib functions.

Font Set Definitions

A font set is defined by a list of fonts, and the actual names of these fonts vary depending on the locale. A mechanism is required to conveniently specify these lists of fonts on a per locale basis, or per application basis.

The actual definition of a font set (that is, the XLFD names of the fonts constituting the font set) can be placed in a font set database. XView refers to this font set database as the font set definition file. When an application is invoked, XView creates an X11 resource manager database internally by reading in the font set definition files.

The locale-specific font set definition files used to create the internal X11 resource manager database are:

1. $OPENWINHOME/lib/locale/<locale>/OW_FONT_SETS/OpenWindows.fs

   This is the system wide font set definition file.

2. <directory>/<locale>/OW_FONT_SETS/<appname>.fs

   `<directory>` is the pathname defined by the application with the `XV_LOCALE_DIR` attribute in the `xv_init()` call. This file is an application-specific file used to add any new font set definitions.

   `<locale>` is the value specified by the attribute `FONT_LOCALE`. If `FONT_LOCALE` is not specified, the `<locale>` defaults to the basic locale of the application.

The internal font set specification database is created using the system font set definition file. If the application-specific font set definition file exists, it is merged into the database. If both files contain definitions for the same font set, the entry in the application-specific file overrides the entry in the system file.

Refer to Appendix C, “Font Set Definitions for syntax of the font set definition file.

Compatibility Issues

The following attributes are for use with locales that require only one font to represent the character set—in other words, the C locale and western European locales (locales that use the ISO Latin-1 character set).

- `FONT_NAME`. Setting this attribute creates a font set object containing the specified font.
- `FONT_INFO`. Querying this attribute returns the pointer to the X structure XFontStruct of the font.
- `FONT_PIXFONT`. This attribute is for SunView compatibility. It returns the pixfont representation of the font.
- `XV_XID`. Querying `XV_XID` of a font set object returns the XID of the font. Use this attribute to obtain the XID of a glyph font.

Do not use these attributes for Asian locales.

6

Portability Issues

To make an XView application more portable, follow these guidelines:

- Do not hard code the name of a font in the application. Use values of `FONT_FAMILY_DEFAULT_FIXEDWIDTH`, `FONT_FAMILY_SERIF`, or `FONT_FAMILY_SANS_SERIF` with the `FONT_FAMILY` attribute. Some font families should be avoided—for example, `FONT_FAMILY_LUCIDA`, which is very specific to the ISO Latin-1 font.
- If a program must use its own font names, be sure to have a private font set definition database file available.
- Not all styles are available in all the supported locales, so an application should limit the use of font styles.
- Do not hard code the font size in the application. Use `FONT_SCALE` instead.
- If a program must define its own font sizes, use a private font set definition database file to specify particular sizes with respect to the scales of small, medium, large, and extra large.

Frames

Multibyte or wide character strings can be used to label frame headers and footers. Table 6-4 lists wide character frame attributes.

Table 6-4 Frame Attributes

Wide Character Attributes

```
FRAME_LABEL_WCS
FRAME_LEFT_FOOTER_WCS
FRAME_RIGHT_FOOTER_WCSS
XV_LABEL_WCS
```

Additionally, the frame package deals with input method by providing an input method status region. Input method is enabled for a frame by default, if the input language specified by `XV_LC_INPUT_LANG` supports an input method. If an input method is enabled for a frame or a frame's subwindow, an input method status region is created automatically. The frame maintains the status region above the footer region.

By default, `WIN_USE_IM` is `TRUE` for frames, and subwindows inherit this value unless specified otherwise during object creation.

History

Table 6-5 lists wide character history attributes.

Table 6-5 History Attributes

Wide Character Attributes

```
HISTORY_ADD_FIXED_ENTRY_WCS
HISTORY_ADD_ROLLING_ENTRY_WCS
HISTORY_LABEL_WCS
HISTORY_NOTIFY_PROC_WCS
HISTORY_VALUE_WCS
```

Icons

Multibyte and wide character strings can be used to label icons. The wide character icon attributes are listed in Table 6-6.

Table 6-6 Icon Attributes

Wide Character Attributes

```
ICON_LABEL_WCS
ICON_TRANSPARENT_LABEL_WCS
XV_LABEL_WCS
```

A locale specific X resource can be specified for the icon's font.

Icon.Font.Name.<locale>
icon.font.name.<locale>

where `<locale>` is specified by `XV_LC_BASIC_LOCALE`. This resource is especially useful to specify fonts of differing point sizes in order to fit localized icon label within the boundaries of an icon.

Menus

Menu items and titles can be specified with wide character or multibyte strings. Table 6-7 lists wide character menu attributes.

Table 6-7 Menu Attributes

Wide Character Attributes
MENU_ACCELERATOR_WCS
MENU_ACTION_ACCELERATOR_WCS
MENU_ACTION_ITEM_WCS
MENU_GEN_PIN_WINDOW_WCS
MENU_GEN_PROC_ITEM_WCS
MENU_GEN_PULLRIGHT_ITEM_WCS
MENU_PULLRIGHT_ITEM_WCS
MENU_STRING_WCS
MENU_STRING_ITEM_WCS
MENU_STRINGS_WCS
MENU_STRINGS_AND_ACCELERATORS_WCS
MENU_TITLE_ITEM_WCS

Notices

Notice messages and buttons handle multibyte and wide character strings. Table 6-8 lists wide character notice attributes.

Table 6-8 Notice Attributes

Wide Character Attributes
NOTICE_BUTTON_WCS
NOTICE_BUTTON_NO_WCS
NOTICE_BUTTON_YES_WCS
NOTICE_MESSAGE_STRING_WCS
NOTICE_MESSAGE_STRINGS_WCS
NOTICE_MESSAGE_STRINGS_ARRAY_PTR_WCS

Panels

All panel items handle wide character or multibyte labels, strings, and values. Panel functions also handle wide character and multibyte strings. Table 6-9 lists wide character panel attributes and functions.

Table 6-9 Panel Attributes and Functions

AttributesFunctions	
PANEL_CHOICE_STRING_WCS	panel_get_value_wcs()
PANEL_CHOICE_STRINGS_WCS	panel_set_value_wcs()
PANEL_ITEM_IC_ACTIVE*	
PANEL_LABEL_STRING_WCS	
PANEL_LIST_INSERT_STRINGS_WCS	
PANEL_LIST_ROW_VALUES_WCS*	
PANEL_LIST_STRING_WCS	
PANEL_LIST_STRINGS_WCS	
PANEL_LIST_TITLE_WCS	
PANEL_MASK_CHAR_WC*	
PANEL_NOTIFY_PROC_WCS	
PANEL_NOTIFY_STRING_WCS	
PANEL_MAX_TICK_STRING_WCSG	
PANEL_MAX_VALUE_STRING_WCS	
PANEL_MIN_TICK_STRING_WCS	
PANEL_MIN_VALUE_STRING_WCS	
PANEL_VALUE_DISPLAY_LENGTH	
PANEL_VALUE_WCS	
PANEL_VALUE_STORED_LENGTH_WCS*	

* See the discussion following this table.

PANEL_VALUE_STORED_LENGTH_WCS

PANEL_VALUE_STORED_LENGTH_WCS sets the storage limit of a panel text item in wide characters. If the storage limit of a panel text item is specified by PANEL_VALUE_STORED_LENGTH_WCS, the characters entered into that panel text item are converted into wide characters to measure against the storage limit. The default value for PANEL_VALUE_STORED_LENGTH_WCS is 80 wide characters.

Usage of this attribute is similar to PANEL_VALUE_STORED_LENGTH. PANEL_VALUE_STORED_LENGTH specifies the storage length of a panel text item in bytes. Therefore, all text input to a panel text item specified by PANEL_VALUE_STORED_LENGTH is converted to multibyte to check against the storage limit.

If PANEL_VALUE_STORED_LENGTH_WCS is set to 10 wide characters, the text item can accommodate 10 Chinese wide characters or 10 ASCII characters. If PANEL_VALUE_STORED_LENGTH is set to 10 bytes, the text item can accommodate 10 ASCII characters, but possibly only 2 Han characters.

Calling xv_get() on PANEL_VALUE_STORED_LENGTH_WCS when the actual attribute set was PANEL_VALUE_STORED_LENGTH returns -1. Similarly, querying the value of PANEL_VALUE_STORED_LENGTH when PANEL_VALUE_STORED_LENGTH_WCS was set returns -1.

PANEL_ITEM_IC_ACTIVE

Occasionally a user wants to disable preedit text input for a particular panel item; for example, a panel numeric text item. Setting PANEL_ITEM_IC_ACTIVE to FALSE temporarily disables the input method for that panel item.

The default value for PANEL_ITEM_IC_ACTIVE is set according to the value of WIN_USE_IM. By default, panel numeric text items are always created with PANEL_ITEM_IC_ACTIVE set to FALSE, since numeric text items only accept ASCII numbers.

Do not use WIN_IC_ACTIVE on panels because it causes conflicts with PANEL_ITEM_IC_ACTIVE.

PANEL_LIST_ROW_VALUES_WCS

PANEL_LIST_ROW_VALUES_WCS offers a high-performance method of obtaining row values and setting row values in the PANEL_LIST. This attribute takes the row number, a pointer to a Panel_list_row_values_wcs array, and a count of the number of rows in the array. Panel_list_row_values_wcs is defined as:

```
typedef struct {
    wchar_t *string_wcs;
    Server_imageglyph;
    Server_imagemask_glyph;
    Xv_fontfont'
    Xv_opaqueclient_data;
    Xv_opaqueextension_data;
    unsignedinactive : 1;
    unsignedselected : 1;
} Panel_list_row_values_wcs;
```

PANEL_MASK_CHAR_WC

PANEL_MASK_CHAR_WC supports wide character masking of panel text values. A panel text item with PANEL_MASK_CHAR_WC or PANEL_MASK_CHAR set causes PANEL_ITEM_IC_ACTIVE to be FALSE. Input method is disabled because preedit text is masked.

Implicit Commit

Implicit commit can be triggered via keyboard and mouse events. “Implicit Commit of Preedit Text” on page 38 lists the keyboard and mouse events that cause implicit commit in panel. Implicit commit can also be triggered programmatically by querying PANEL_VALUE and PANEL_VALUE_WCS while input method is active.

Pathnames

Table 6-10 lists wide character pathname attributes.

Table 6-10 Pathname Attributes

Wide Character Attributes
PATH_LAST_VALIDATED_WCS
PATH_RELATIVE_TO_WCS

Resources

The XView database functions are not part of an XView package, but they make it possible to handle X resources.

Table 6-11 lists functions related to locale-specific resources.

Table 6-11 Resource Functions

Functions
defaults_set_locale()*
defaults_get_locale()

* See the discussion following this table.

In the internationalized and localized environment, the user often needs to specify resources for a particular locale. For example, the user may want to specify a special font for the Korean locale, but a different font for the Japanese locale. The `~/.Xdefaults` file and the corresponding server resources are usually shared among locales, so an optional syntax is added to support the requirement of locale-specific resources in the XView environment:

<original_resource_name>{.<locale>}

`original_resource_name` is a resource name, such as “Font.Name”, and `<locale>` is a locale name, such as “`zh`” for simplified Chinese.

The locale-specific resource overrides the original resource.

`defaults_set_locale()` allows an application to support the aforementioned optional syntax:

```
void defaults_set_locale(locale, locale_attr)
char *locale;/* locale value, if known */
int locale_attr; /*a locale category, i.e., basic locale */
```

Only one parameter needs to be specified: `locale` or the `locale_attr`.

The first call to `defaults_set_locale()` with a locale or a locale category activates the locale-specific resources lookup mechanism. Another call to `defaults_set_locale()` with `NULL` terminates the locale-specific resources lookup mechanism. XView does not search for any locale-specific resources before `defaults_set_locale()` is called in an application. There is a performance penalty associated with this syntax, and a locale must be specified in the application. For example:

```
defaults_set_locale(NULL, XV_LC_INPUT_LANG);
value = defaults_get_string("my_input_style",
                "my_input_style",
                "my_favorite");
defaults_set_locale(NULL, XV_NULL);
```

First, `defaults_set_locale(NULL, XV_LC_INPUT_LANG)` sets the locale for the search to be the locale of the input language. Next, `defaults_get_strings()` looks for `my_input_style.<locale>`. If `my_input_style.<locale>` cannot be found, `defaults_get_strings()` looks for the non-locale specific resource `my_input_style`. Finally, `defaults_set_locale(NULL, XV_NULL)` terminates the locale-specific resources lookup mechanism.

XView automatically loads the X resource database files at startup. A locale-dependent X resource database file is also loaded:

$OPENWINHOME/lib/locale/<basiclocale>/xview/defaults

The purpose of this file is to set locale-dependent resources, such as icon font name. This locale-dependent defaults file has the lowest priority among X resource database files loaded in Asian locales. (Other X resource database files can override the contents of this locale-dependent defaults file—for example, a user's `~/.Xdefaults` or `~/.OWdefaults` file.)

Table 6-12 lists resources that have been modified so that they can be specified in a locale-sensitive manner. Their names vary depending on the basic locale.

Table 6-12 Locale-Sensitive Resources

Resource
font.name.<basiclocale>
Font.Name.<basiclocale>
icon.font.name.<basiclocale>
Icon.Font.Name.<basiclocale>
OpenWindows.BoldFont.<basiclocale>
OpenWindows.ImPreeditStyle.<basiclocale>
OpenWindows.ImStatusStyle.<basiclocale>
OpenWindows.MonospaceFont.<basiclocale>
OpenWindows.RegularFont.<basiclocale>
text.extrasMenufilename.<displaylang>*
xview.needIm.<basiclocale>*
xview.characterSet.<basiclocale>*

* See the discussion following this table.

For font resources, replace `<basiclocale>` with the value of the basic locale. The default value for the font resources is lucidasans-12 in the C and ISO Latin-1 locales. For other locales, refer to this file:

$OPENWINHOME/lib/locale/<basiclocale>/OW_FONT_SETS/OpenWindows.fs

For `text.extrasMenufilename`, replace `<displaylang>` with the value of the display language. Default values:

- `$OPENWINHOME/lib/locale/<locale>/xview/.text_extras_menu` (where `<locale>` is the value of the display language). or
- /usr/lib/.text_extras_menu

Refer to “Text Subwindows” on page 63 for further information.

The locale-specific resource `xview.needIM` specifies whether an input method is necessary for the input language. It is typically specified in `$OPENWINHOME/lib/locale/<locale>/xview/defaults`; therefore, users should not need to set this in their `~/.Xdefaults` file.

The locale-specific resource `xview.characterSet` describes the type of character set associated with a particular locale. For instance, the following would be specified for western European locales that are based on the ISO Latin-1 character set:

`xview.characterSet.<locale>:iso_8859_1`

Since Asian locales typically use a character set that is unique for each locale, the following would be specified:

xview.characterSet.ja: ja
xview.characterSet.ko: ko
xview.characterSet.zh: zh
xview.characterSet.zh_TW: zh_TW

The `xview.characterSet` resource is typically specified in `$OPENWINHOME/lib/locale/<locale>/xview/defaults`; therefore, users should not need to set this in their `~/.Xdefaults` file.

Specifying `xview.characterSet` allows for better interoperability between locales that share the same character set. For example, since German (`de`) and French (`fr`) locales both use the ISO Latin-1 character set, it would be valid to have `XV_LC_BASIC_LOCALE` set to `fr` and `XV_LC_DISPLAY_LANG` set to `de`. Such interoperability is not possible with Asian locales because they each use different character sets.

Selections

Table 6-13 lists wide character selection attributes.

Table 6-13 Selection Service Attributes

Wide Character Attributes

`SELN_REQ_CONTENTS_WCS*`
`SELN_REQ_CHARSIZE*`
`SELN_REQ_FIRST_WC*`
`SELN_REQ_LAST_WC*`

* See the discussion following this table.

`SELN_REQ_CHARSIZE` returns the number of characters in a selection buffer.

`SELN_REQ_FIRST` and `SELN_REQ_LAST` provide the indices of the first and last selected characters in bytes. `SELN_REQ_FIRST_WC` and `SELN_REQ_LAST_WC` provide the indices of the first and last selected character in characters.

When you use the attribute SELN_REQ_CONTENTS_WCS, SELN_REQ_FIRST_WC, or SELN_REQ_LAST_WC in selection_query(), selection_ask(), or selection_init_request(), also use the SELN_REQ_CHARSIZE attribute. The following example first determines if the wide character attribute is needed and then gets the selection.

```
Xv_Serverserver;
Seln_holder*holder;
Seln_result(*reader)();
char    *context;
Seln_resultquery_result;

query_result =
    selection_query(server, holder, reader, context,
           SELN_REQ_CHARSIZE, NULL,
           NULL);

if (query_result != SELN_SUCCESS) {

    /* An XView client running in C locale */

    query_result =
        selection_query(server, holder, reader, context,
        SELN_REQ_BYTESIZE, NULL,
        SELN_REQ_CONTENTS_ASCII, NULL,
        NULL);
}
else {
    /* An XView client running in an Asian locale */

    query_result =
        selection_query(server, holder, reader, context,
        SELN_REQ_CHARSIZE, NULL,
        SELN_REQ_CONTENTS_WCS, NULL,
        NULL);
}
```

Server Images

SERVER_IMAGE_BITMAP_FILE_WCS supports wide character filenames.

Text Subwindows

This section describes attributes and functions for text subwindows, programming considerations, and miscellaneous tips and information.

Multibyte and Wide Character API

This section discusses text subwindow attributes and functions that process wide character strings and character-based indices (positioning in a text subwindow).

Buffer, Index, or Length Text Subwindow API

The wide character attributes and functions in Table 6-14 may have been added because the attribute or function:

- Takes a wide character buffer as argument.
- Processes a character-based index as an argument or return value.
- Returns the number of inserted or deleted characters (not bytes).
- Gets the number of characters (not bytes) in a text subwindow's contents.

Table 6-14 Text Subwindow APIs that Take Buffer, Index, or Length

Wide Character Attributes	Wide Character Functions
`TEXTSW_CONTENTS_WCS`	`textsw_add_mark_wc()`
`TEXTSW_FIRST_WC`	`textsw_delete_wcs()`
`TEXTSW_INSERTION_POINT_WC`	`textsw_edit_wcs())`
`TEXTSW_LENGTH_WC`	`textsw_erase_wcs()`
	`textsw_find_wcs()`
	`textsw_find_mark_wc()`
	`textsw_index_for_file_line_wc()`
	`textsw_insert_wcs()`
	`textsw_match_wcs()`
	`textsw_normalize_view_wc()`
	`textsw_possibly_normalize_wc()`
	`textsw_replace_wcs()`
	`textsw_set_selection_wcs()`

The arguments and return values for this API mean different things depending on whether the multibyte or wide character API is used, as indicated in Table 6-15.

Table 6-15 Differences Between Multibyte and Wide Character API

Argument/Return Value	Wide Character API	Multibyte API
character buffer	wchar_t *	char *
index (Textsw_index)	character base	byte base
buffer length	character base	byte base
inserted or deleted length	character base	byte base
length of textsw's contents	character base	byte base

Text Subwindow Filename API

Table 6-16 lists attributes and functions for wide character filenames. They work the same as those for multibyte except that the filename string is specified in wide character format (`wchar_t *`). The attributes `TEXTSW_ACTION_*` are used to make notify procedures for a text subwindow.

Table 6-16 Text Subwindow Filename Attributes and Functions

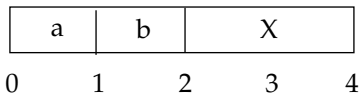
Wide Character Attributes	Wide Character Function
TEXTSW_ACTION_CHANGED_DIRECTORY_WCS	textsw_append_file_name_wcs()
TEXTSW_ACTION_EDITED_FILE_WCS	textsw_store_file_wcs()
TEXTSW_ACTION_LOADED_FILE_WCS	
TEXTSW_FILE_WCS	
TEXTSW_FILE_CONTENTS_WCS	
TEXTSW_INSERT_FROM_FILE_WCS	

Programming Considerations for Text Subwindow Multibyte API

The following information points out special factors to consider: index adjustments, invalid data and buffer length adjustments, and extra processing tasks that influence performance.

Index Adjustments

When a specified index points to the middle (portion) of a multibyte character, the index is adjusted to the front of the character. For example, if a text subwindow contains “abX” (‘a’ and ‘b’ are ASCII characters, ‘X’ is a multibyte character consisting of two bytes) and the index is specified as 3, the index is set to 2 because index 3 points to the middle of a multibyte character.



Invalid Data and Buffer Length Adjustments

Table 6-17 lists multibyte APIs that take a buffer as an argument.

Table 6-17 Multibyte APIs that Take a Buffer as an Argument

Multibyte Attributes	Multibyte Functions

`TEXTSW_CONTENTS`	`textsw_find_bytes()`
	`textsw_match_bytes()`
	`textsw_replace_bytes()`
	`textsw_insert()`

If the API takes buffer length as an argument, the buffer length is specified in bytes.

These APIs ignore the current character and stop processing the rest of the string when the following conditions occur:

- The function finds an invalid character in the specified buffer.
- The specified buffer length includes a portion of a multibyte character, but not the entire character.

For example, a buffer contains “ABC”, where each character is a multibyte character and the buffer length is five bytes. In the Japanese locale, only “AB” is processed because a multibyte character in the Japanese locale consists of two bytes. In traditional Chinese (`zh_TW` locale), only “A” is processed because a multibyte character in that locale uses four bytes.

The following example, which reads characters from a file and inserts the characters into a text subwindow, illustrates some of the issues involved.

```
#define READ_SIZE512
charbuf[READ_SIZE];
int read_cnt;
Textsw_index   ret_val;

while (1) {
    read_cnt = read(file, buf, READ_SIZE);
    switch (read_cnt) {
        case 0:
            break;/* read all data of file */
        case -1:
            break;/* read error */
        default:
            ret_val = textsw_insert(textsw, buf, read_cnt);
            if (ret_val == 0)
                /* memory allocation failure */
            else if ((int)ret_val != read_cnt)
                /* insertion unsuccessful */
            else
                /* insertion successful */
            break;
    }
}
```

If the data in the file is valid and `ret_val` is not equal to `read_cnt`, the last bytes inserted in `buf` are only a portion of a multibyte character. In this case, you may need to adjust the read pointer in the file to the front of the uninserted bytes using the `seek()` system call, or you may need to keep the extra bytes and pass them into `textsw_insert()` together with the next data read from the file.

Text Subwindow Performance

Note – This section applies only to Asian (multibyte) locales.

The wide character API is recommended when programming with `textsw` indices or buffers. The wide character API provides better performance as well as ease of programming. XView `textsw` handles all strings and indices internally in wide characters. The index or buffer adjustment problems mentioned above can be avoided with the wide character API.

The multibyte `textsw` API must perform the following extra tasks each time a string or index is processed: conversions for string, conversion for index, and calculation for length.

- *Conversion for String*
  A multibyte API that takes a buffer as an argument, must convert the multibyte string to wide character before the string can be processed. This is because text subwindows internally handle strings in wide character. In addition, when the API returns the contents of a text subwindow, the contents are converted to multibyte from wide character. `textsw_insert()` and `textsw_find_bytes()` are examples of the conversion from multibyte to wide character. `TEXTSW_CONTENTS` used with `xv_get()` is an example of the conversion from wide character to multibyte character.

- *Conversion for Index*
  A multibyte API that takes an index as an argument must convert the byte-based index to a character-based index at the beginning of the process. The text subwindow internally handles the index in characters, not in bytes. Also, when the API returns an index, the index is converted to a byte-based index from a character-based index. Note that when the index value is `TEXTSW_INFINITY`, it is not converted. The call below suffers no penalty:

  ```
  xv_set(textsw,TEXTSW_INSERTION_POINT, TEXTSW_INFINITY, 0);
  ```

  Note also that retrieving the next read position (`TEXTSW_CONTENTS` with `xv_get())` does not suffer a penalty for converting the index.

- *Calculation for Length*
  The multibyte attribute `TEXTSW_LENGTH` returns the length of the text subwindow contents in bytes. Some multibyte APIs return the length of the deleted wide character string in bytes (for example, `textsw_delete` and `textsw_edit`). Calculating the number of bytes of a wide character string also causes a performance penalty. Note that calculating the number of bytes of the multibyte string inserted by `textsw_insert()` does not cause a performance penalty.

If the same multibyte string is repeatedly passed to the multibyte API, the application should internally convert the multibyte string to a wide character string and use the wide character API. A text subwindow application should convert to the wide character API if conversion only required name changes of the attribute or function. For example:

- When checking whether a text subwindow is empty or not, the following code can be changed to use `TEXTSW_LENGTH_WC`:

```
if (xv_get(textsw, TEXTSW_LENGTH) == 0)
```

- When repositioning text so that the character at the current insertion point is visible and at the top of the subwindow, you normally write:

```
textsw_normalize_view(textsw,
xv_get(textsw, TEXTSW_INSERTION_POINT))
```

  This can be replaced by the wide character API as follows:

```
textsw_normalize_view_wc(textsw,
xv_get(textsw, TEXTSW_INSERTION_POINT_WC));
```

- When deleting all the contents using `textsw_delete()`:

```
textsw_delete(textsw, 0, TEXTSW_INFINITY);
```

In this case, the argument indices are not converted, but the number of deleted bytes is returned. It should be replaced by `textsw_delete_wcs()`.

Other Text Subwindow Information

Invalid Data in Files

The text subwindow checks each file being loaded or included for invalid characters. If a file contains invalid characters, the file is still loaded or included but the invalid characters are skipped over. A notice window will be displayed to inform the user that this file contains invalid characters. The invalid characters will not be stored when the contents of the text subwindow are saved. The contents of the text subwindow should not be saved if the invalid characters are to remain.

If text is inserted into a text subwindow from a buffer by using the multibyte API, and if the text contains invalid data, the invalid characters are not skipped over. Refer to "Invalid Data and Buffer Length Adjustments" on page 65 for details.

An *invalid character* is defined to be any character that does not exist in the code set of the current locale. For example, an ISO Latin-1 character in the Korean locale.

Creation of a Temporary File

In Asian locales (for example, `ja`, `ko`, `zh`, and `zh_TW`), the text subwindow creates a temporary file by using the `tempnam()` function when loading or saving a file.[1]

This temporary file is generally created under the `/tmp` directory without setting an environment variable `TMPDIR`. Refer to the `tempnam(3S)` manpage for further information. If the `/tmp` directory is out of disk space and the temporary file cannot be created, the operation fails and a notice window advising of the failure is displayed. To avoid this problem, specify the environment variable `TMPDIR` to a directory with adequate disk space; then restart a program that uses the text subwindow.

Implicit Commit

Implicit commit can be triggered by keyboard and mouse events. “Implicit Commit of Preedit Text” on page 38 lists the events that cause implicit commit in text subwindows.

The actions listed in Table 6-18 also cause implicit commit in text subwindows. Some of these actions are initiated by the use of certain text subwindow APIs; others are caused by operations from the Text Pane menu.

Table 6-18 Implicit Commit Actions and Corresponding API Examples

Implicit Commit Action	Example API that Triggers Action
`Move the caret`	`TEXTSW_INSERTION_POINT`
`Retrieve the contents`	`TEXTSW_CONTENTS`, `textsw_store_file()`

1. The temporary file is invisible because it is removed with `unlink (1M)` immediately after its creation. The file resource is not freed while the process that uses the text subwindow has access to the temporary file.

Change the contents	textsw_delete(), textsw_replace_bytes()
Edit mode becomes read only	TEXTSW_READ_ONLY set TRUE
Language input mode becomes inactive	WIN_IC_ACTIVE set FALSE

WIN_USE_IM *and* TEXTSW_READ_ONLY

If the text subwindow is created in read-only mode and will never need an input method, WIN_USE_IM should be set to FALSE. The FALSE setting prevents the creation of an unnecessary input context (IC). Otherwise, an IC is created, even for a read-only text subwindow, because the read-only mode may later change to edit mode, either explicitly (by setting TEXTSW_READ_ONLY to FALSE) or implicitly (by loading a file).

```
xv_create(frame, TEXTSW,
    WIN_USE_IM,FALSE,
    TEXTSW_READ_ONLY,TRUE,
    NULL);
```

Text Subwindow Extras Menu

Table 6-19 lists the search path and priority to find the text subwindow Extras Menu file.

Table 6-19 Extras Menu Search Path

Priority/File Location	Search Path
1 X defaults	text.extrasMenuFilename.<locale>
2 Environment variable	$(EXTRASMENU).<locale>
3 Home directory	$(HOME)/.text_extras_menu.<locale>
4 Locale-sensitive system default	$OPENWINHOME/lib/locale/<locale>/xview\ /.text_extras_menu

```
5 Fallback to SunView (/usr/lib/.text_extras_menu)
```

In priorities 1, 2, and 3, `<locale>` is the locale of the display language. If the Extras Menu file with the locale suffix does not exist, the Extras Menu file without the locale suffix is used. For example, if the definition in the `.Xdefaults` file is:

text.extrasMenuFilename: /tmp/my_extas_menu

and locale is `ja` and there are `my_extras_menu` and `my_extras_menu.ja` in `/tmp`, then use `my_extras_menu.ja`. If there is no `my_extras_menu.ja` in `/tmp`, use `my_extras_menu`.

In priority 4, if there is no `.text_extras_menu` under `$OPENWINHOME/lib/locale/<locale>/xview`, the C locale version is used.

TTY Subwindows

Table 6-20 lists wide character tty subwindow functions.

Table 6-20 TTY Subwindow Functions

Wide Character Functions
`ttysw_input_wcs()`
`ttysw_output_wcs()`

Windows: Handling Input

The window package is a superclass of many other packages. Panel, canvas, text subwindow, and tty subwindows are all subclasses of window. The extensions to the window package consist mainly of attributes that allow input handling for Asian locales through the use of an input method. Attributes are provided to enable and disable the input method for a given window. In addition, many attributes give you the flexibility to replace the input method provided by SunSoft with your own customized user interface. Table 6-21 lists window attributes.

Table 6-21 Window Attributes

Attributes

WIN_ERROR_MSG_WCS
WIN_IC*
WIN_IC_ACTIVE*
WIN_IC_COMMIT_STRING*
WIN_IC_COMMIT_STRING_WCS*
WIN_IC_CONVERSION*
WIN_IC_PREEDIT_CARET*
WIN_IC_PREEDIT_DONE*
WIN_IC_PREEDIT_DRAW*
WIN_IC_PREEDIT_START*
WIN_IC_RESET*
WIN_IC_STATUS_DONE*
WIN_IC_STATUS_DRAW*
WIN_IC_STATUS_START*
WIN_USE_IM*
WIN_X_IM_STYLE_MASK*
XV_IM*
XV_IM_STYLES*

* See the discussion following this table.

Enabling the Input Method

The WIN_USE_IM attribute enables or disables the input method for a given window. If the locale specified by XV_LC_INPUT_LANG supports an input method, and if the xview.needIm resource and the WIN_USE_IM attribute are TRUE, the input method will be enabled. If a subwindow does not require an input method, create the object with WIN_USE_IM set to FALSE. This avoids the overhead of creating unnecessary input contexts.

When used with base frames or command frames, setting WIN_USE_IM to FALSE disables the input method and removes the status region at the bottom of the frame. Subwindows inherit the WIN_USE_IM value specified by their parent frame. WIN_USE_IM can be set explicitly for each subwindow. WIN_USE_IM can be specified only during the creation of the object.

The example below demonstrates how to create a panel with the input method disabled. Note that the panel inherits the frame's WIN_USE_IM value.

```
#include <xview/xview.h>
#include <xview/frame.h>
#include <xview/panel.h>

Frameframe;
Panelpanel;

main (argc, argv)
int argc;
char*argv[];
{

    xv_init(XV_INIT_ARGC_PTR_ARGV, &argc, argv,
        XV_USE_LOCALE, TRUE,
        NULL);

    frame = xv_create(NULL, FRAME,
        WIN_USE_IM, FALSE,
        NULL);

    panel = xv_create(frame, PANEL,
        NULL);

    xv_create(panel, PANEL_TEXT
        PANEL_VALUE, dgettext("example_message_domain", "ASCII File Name"),
        NULL);

    window_fit(panel);
    window_fit(frame);

    xv_main_loop(frame);

}
```

There may be instances when you want to temporarily disable the input method after WIN_USE_IM has been enabled. You can accomplish this by setting WIN_IC_ACTIVE to FALSE.

Note – Setting `WIN_IC_ACTIVE` to `FALSE` to disable the input method temporarily is not equivalent to creating an object with `WIN_USE_IM` set to `FALSE`. Although setting `WIN_IC_ACTIVE` allows you to temporarily disable input method, it does not free input context s associated with the window, nor does it prevent input contexts from being created. If the window does not require an input method, create the object with `WIN_USE_IM` set to `FALSE`.

Input Method and Input Context

The Xlib `XIM` handle can be retrieved using the `XV_IM` attribute. This is useful if you want to query information about the input method via Xlib functions. If `XV_IM` returns `NULL`, no IM connection was made.

An input context (IC) can be specified by the application with a direct Xlib call to `XCreateIC()`. By default, each window that specifies `WIN_USE_IM` to be `TRUE` has an IC associated with it. There are also default preedit and status callbacks registered for each window's IC.

A window's default IC can be retrieved or set by calling `xv_get()` or `xv_set()` on the `WIN_IC`. The window package manages both the creation and destruction of the default IC. It is not necessary to call `XDestroyIC()` unless you are creating your own IC.

Choosing Input Style

The attribute `WIN_X_IM_STYLE_MASK` specifies the input style mask for the input method. The mask specifies preedit and status styles.

If `WIN_USE_IM` is set to `TRUE`, `WIN_X_IM_STYLE_MASK` affects the IC associated with the window. The input style is determined by the style specified by the application or user, by the locale setting, and by the IM server. (See "Input Method Styles" on page 34 for details about style determination.) If your application relies on a particular IM style, be sure to query `XV_IM_STYLES` to determine which styles are available. Do this before creating a window with `WIN_X_IM_STYLE_MASK`. `WIN_X_IM_STYLE_MASK` can be set only during `xv_create()`.

```
/* Queries the supported input styles */
Xv_Server server;
XIMStyles styles;

server  =  xv_init (XV_INIT_ARGV, &argc, argv,
                    XV_USE_LOCALE, TRUE,
                    NULL);
styles  =  xv_get(server, XV_IM_STYLES);

/* Based on style information we can then decide
   what style the window should be created with */

...

/* Creates panel with over-the-spot and im-displays-in-client
input style */
xv_create (panel, WIN_USE_IM, TRUE,
                    WIN_X_IM_STYLE_MASK,
        XIMPreeditPosiition | XIMStatusArea,
                    NULL);
```

Customizing Implicit Commit

As described in "Implicit Commit of Preedit Text" on page 38 certain mouse and keyboard actions can automatically commit preedit text without the user having to enter a commit key sequence. You may want to customize implicit commit actions.

The attributes described in the example below can be used with each other to commit the preedit string and return the preedit string value. `WIN_IC_RESET` turns input method conversion off and commits the preedit string by clearing any pending input for the input context associated with the given window.

`xv_get()` of `WIN_IC_COMMIT_STRING` or `WIN_IC_COMMIT_STRING_WCS` returns the committed string in either multibyte or wide character format. `WIN_IC_CONVERSION` can be set to `TRUE` to turn input method conversion back on, after `WIN_IC_RESET` has turned conversion off.

```
#include <xview/xview.h>
#include <xview/frame.h>
#include <xview/canvas.h>

Frameframe;
Canvascanvas;

voidcanvas_event_proc();

main (argc, argv)
int argc;
char*argv[];
{

    xv_init(XV_INIT_ARGC_PTR_ARGV, &argc, argv,
        XV_USE_LOCALE, TRUE,
        NULL);

    frame = xv_create(NULL, FRAME,
        WIN_USE_IM, TRUE,/* default */
        NULL);

    canvas = xv_create(frame, CANVAS,
        NULL);

    xv_set(canvas_paint_window(canvas),
        WIN_CONSUME_EVENTS,
            WIN_NO_EVENTS,
            WIN_ASCII_EVENTS, KBD_USE, KBD_DONE,
            LOC_DRAG, LOC_WINENTER, LOC_WINEXIT, WIN_MOUSE_BUTTONS,
            NULL,
        WIN_EVENT_PROC, canvas_event_proc,
            NULL);

    xv_main_loop(frame);
}

\*  CODE EXAMPLE CONTINUED ON NEXT PAGE!!! *\
```

```
void
canvas_event_proc(window, event, arg)
Xv_Window       window;
Event           *event;
Notify_arg      arg;
{

        ...


    switch (event_action(event)) {

        ...

       /*
        * Middle mouse button will trigger implicit commit.
        */
       case ACTION_ADJUST:
       case MS_MIDDLE:

        /* Commit the preedit string */
        xv_set(window, WIN_IC_RESET, TRUE, NULL);

        /* Retrieve the committed text */
        committed_string = (wchar_t *)xv_get(canvas,
            WIN_IC_COMMIT_STRING_WCS);

        /* Turn conversion back on, after being turned off
         * with WIN_IC_RESET. */
        xv_set(canvas, WIN_IC_CONVERSION, TRUE, NULL);
        break;

       default:
        return;
    }

    ...

}
```

6

Customizing Input Method Callbacks

The input method user interface is implemented through a number of callback procedures for the preedit and status regions when on-the-spot and client-displays styles are used. You can, however, choose to use you own customized callback procedures instead of those provided in the current XView release by using the following attributes to set callbacks.

- Preedit region:
  - `WIN_IC_PREEDIT_START`
  - `WIN_IC_PREEDIT_DRAW`
  - `WIN_IC_PREEDIT_CARET`
  - `WIN_IC_PREEDIT_DONE`
- Status region:
  - `WIN_IC_STATUS_START`
  - `WIN_IC_STATUS_DRAW`
  - `WIN_IC_STATUS_DONE`

The following is an example of interposing the preedit and status callbacks.

```
Framebase_frame;
Textswtextsw;
Panelpanel;
XIMProcmy_text_start, my_text_draw, my_text_end;
XIMProcmy_panel_start, my_panel_draw, my_panel_end;

xv_init(....);
base_frame = xv_create(NULL, FRAME, 0);
textsw = xv_create(base_frame, TEXTSW, 0);
panel = xv_create(base_frame, PANEL, 0);

xv_set(textsw,
   WIN_IC_PREEDIT_START, my_text_start,
     (XPointer) textsw,
   WIN_IC_PREEDIT_DRAW, my_text_draw,
     (XPointer) textsw,
   WIN_IC_PREEDIT_DONE, my_text_end,
     (XPointer) textsw,
   0);
xv_set(panel,
   WIN_IC_PREEDIT_START, my_panel_start,
        (XPointer) panel,
   WIN_IC_PREEDIT_DRAW, my_panel_draw,
        (XPointer) panel,
   WIN_IC_PREEDIT_DONE, my_panel_end,
        (XPointer) panel,
   0);
```

6