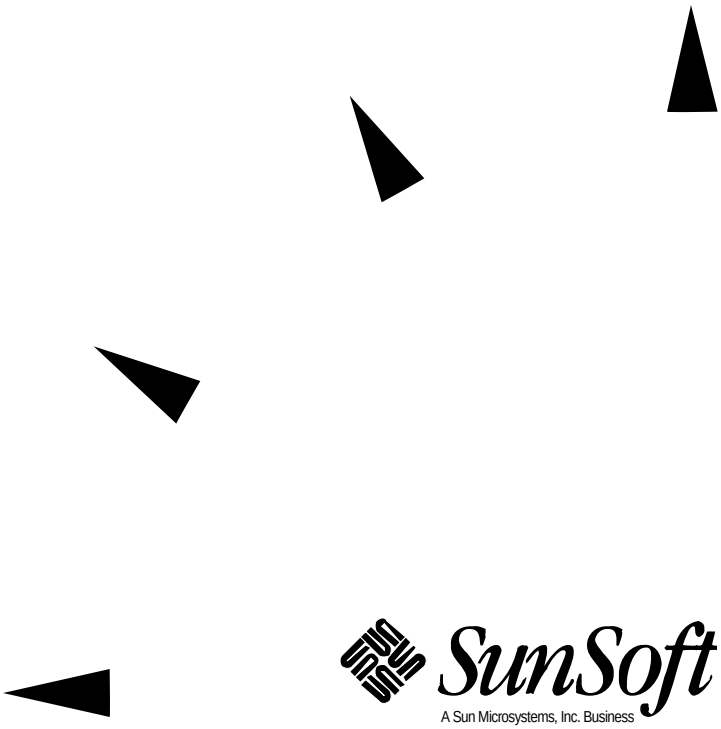
XView Developer's Notes



2550 Garcia Avenue
Mountain View, CA 94043
U.S.A.

SunSoft
A Sun Microsystems, Inc. Business

Please
Recycle

Adobe PostScript

Contents

Figures

Tables

Code Samples

Preface

*XView Developer's Note*s provides XView developer information not present in the O'Reilly XView documentation set. This manual contains three parts:

- Motif interoperability and O'Reilly Supplements and Corrections (Part I, chapters 1 and 2)
- Internationalizing XView applications (Part II, chapters 3 through 6 and appendices A through C)
- Release Information (appendix D)

Part I discusses interoperability issues between the current XView release and Motif. It also contains corrections and supplements to the O'Reilly programmer (Third Edition) and reference documentation for XView Version 3. Part II enhances, and builds upon, XView internationalization information present in the O'Reilly documentation set and in other SunSoft manuals. Part III contains information specific to this release, such as new features.

Who Should Use This Book

This manual is intended for XView application developers, who are proficient in the C programming language and in XView programming.

Refer to the following O'Reilly and Associates manuals for help with the XView toolkit:

- *XView Programming Manual, Third Edition, for XView Version 3*.2 (Volume 7) by Dan Heller

- *XView Reference Manual for XView Version 3*.2 (Companion to Volume 7)
- *Xlib Programming Manual* for Version 11 (Volume 1) by Adrian Nye
- *Xlib Reference Manual* for Version 11 (Volume 2)

You can order these books through Sun Express or buy them at your local bookstore.

This manual assumes that you have access to the *Developer's Guide to Internationalization*.

Before You Read This Book

Read Appendix D, "XView Release Notes," for important information about this release.

Check the following manuals for any corrections or updates to information in this manual:

- *SPARC: Installing Solaris Software*
- *x86: Installing Solaris Software*
- *Software Developer Kit Open Issues and Late-Breaking News*
- *Software Developer Kit Introduction*
- *Software Developer Kit Installation Guide*

Internationalizing Applications

Before you start, be sure to prepare properly:

- Read the *Developer's Guide to Internationalization*.
- For Asian locales, install the applicable Asian language environment:
  - KLE for Korean
  - JFP for Japanese
  - HLE for traditional Chinese
  - CLE for simplified Chinese

Refer to the Internationalization chapter in the *XView Programming Manual* for information about attributes and functions that work well with English and western European languages. Part II of this manual describes new and modified attributes and functions that let your applications work in Asian locales as well.

Meta and Help Key Alternatives

If your keyboard does not have a Meta key, use the Control-Alt key sequence (pressed simultaneously) to perform the same functionality as Meta. If you do not want Control-Alt to serve as the Meta function, add the OpenWindows.CtrlAltMetaKey resource to your .Xdefaults file and set it to False.

The F1 function key serves as the Help key, if your keyboard does not have a Help key.

Typographical Conventions

Attributes, procedures, macros, and anything resembling C code are all set in `Courier`, using the following C conventions:

- Procedures are lowercase and are followed by parentheses, as in:

  `xv_get()`

- Macros are all uppercase and are followed by parentheses, as in the following example, except where they are lowercase in source code:

  OPENWIN_EACH_VIEW()

- Attributes are all uppercase but are not followed by parentheses, as in:

  CANVAS_MIN_PAINT_HEIGHT

Representations of anything that might appear on your screen are set in `Courier`. In addition, code examples are set in boxes to set them off from the surrounding text.

Further Documentation

American National Standard for Information Systems - Programming Language C, ANSI X3.159-1989, Published by American National Standard Institute.

OPEN LOOK GUI Functional Specification, International Extension, Unix International, Internationalization Working Group.

IEEE Standard Portable Operating System Interface for Computer Environments, POSIX 1003.1, IEEE Std 1003.1, Published by The Institute of Electrical and Electronics Engineering, Inc. ISBN 1-55937-003-3.

Xlib - C Language X Interface MIT X Consortium Standard X Version 11, Release 5.

X Window System C Library and Protocol Reference, Release 5 Version, Robert W. Scheifler & James Getty, 1992, Digital Press

X Programmers Supplement for Release 5, David Flanagan, 1991, O'Reilly and Associates

Japanese Input User's Guide, SunSoft

JFP User's Guide, SunSoft

JFP Programmer's Guide, SunSoft

Asian Solaris 2.4 User's Guide, SunSoft

Part 1 — Miscellaneous X View Issues

Motif and XView Interoperability 1

The XView toolkit included in Solaris releases implements the OPEN LOOK® "look and feel." Because OPEN LOOK and Motif® standards differ, XView clients (applications) do not interoperate properly with Motif clients and the Motif window manager in some cases. This chapter describes interoperability issues for the current Solaris release.

Note – The openwin window system start up script includes a new option that allows the user to start the Motif Window Manager instead of the OPEN LOOK Window Manager.

Motif and XView Client Interoperability

Motif and XView clients experience interoperability problems in two areas: selections and drag and drop.

Selections

Motif supports three modes of selection: primary, secondary, and clipboard. XView supports these modes, also, but uses a different mechanism for primary and secondary selections. Motif and XView, then, do not interoperate for these two types of selections. The Motif primary copy operation cannot copy into an XView client. Motif clients cannot perform a quick copy/paste operation into an XView client, or vice versa.

In the C locale, a user can use cut, copy, and paste properly between Motif and XView clients. This functionality might not work in other locales.

Drag and Drop

Motif and XView use different internal protocols and user interface paradigms for drag and drop. Thus an end user cannot drag and drop data between Motif and XView applications.

Motif Window Manager and XView Client Interoperability

The Motif window manager and XView clients experience interoperability problems in four areas: XView clients with two base windows, window decoration, XView Text Editor, and Focus Follows Mouse for XView Clients.

XView Clients with Two Base Windows

If an XView application in a Motif environment (with the Motif Window Manager running) has two base windows, and the user quits one window, then the entire application quits.

Window Decoration

XView applications need to communicate with the Motif Window Manager to control their window decorations. The XView toolkit will implement the Motif Close menu button, Resize corners, and the Window title.

XView Text Editor Client

If a user is editing with Text Editor in a Motif environment, and decides to quit the application, Text Editor *does not* produce a notice asking if he wants to perform a save operation first.

Focus Follows Mouse

When a user starts the Motif Window Manager with `FocusPolicy -PointerLocation`, Motif clients get the focus immediately. XView clients *do not* get the focus; the user must click on the client to get the focus.

1

O'Reilly Corrections and Supplements

This appendix contains corrections to the XView O'Reilly documentation, as well as some supplementary XView material.

Corrections to the XView Programming Manual

This section contains corrections to the XView Programming Manual (Volume 7), Third Edition, from O'Reilly and Associates. Unless noted, the corrections are for the XView version 3 edition of the manual.

`long_seln.c`

The `long_seln.c` program (example A-3) contains an error. The line

```
char *seln_bufs[3]; /* contents of each of the three selections */
```

should be replaced by:

```
char *seln_bufs[6]; /* contents of the three selections, but
                       allow room for all six types of selections*/
```

The program in `$OPENWINHOME/share/src/xview/examples/seln_svc` contains the corrected code.

PANEL_EVENT_PROC

The `xv_set` expression in Section 7.19.8

```
xv_set(panel, PANEL_EVENT_PROC, my_event_proc, NULL)
```

incorrectly sets the `PANEL_EVENT_PROC` on a `panel`. It can be set only on a `Panel_item`.

notify_next_event_func

The Third Edition O'Reilly programmer documentation incorrectly states that `notify_next_event_func` takes the same arguments as `my_frame_interposer` (see example 20-5). (Note that this error also occurs in the function declaration on page 162 of the reference manual.)

Using the function with these parameters leads to a compiler warning:

```
line # : warning: improper pointer/integer combination: arg #2
```

The `notify_next_event_func` declaration should read:

```
Notify_value
notify_next_event_func(client, event, arg, type)
       Notify_client  client;
       Notify_event   event;
       Notify_arg     arg;
       Notify_event_typetype;
```

notice.c

This correction refers to the XView 3.2 Programming Manual.

The `notice.c` program (example 12.6) incorrectly uses the NOTICE package. To correct this, declare `result` as type `static int` (instead of `int`) in `my_notify_proc`.

Corrections to the XView Reference Manual

This section contains corrections to the XView Reference Manual for XView Version 3, Companion to Volume 7, from O'Reilly and Associates.

CMS_COLOR_COUNT

You cannot use `CMS_COLOR_COUNT` to retrieve a subset of the CMS's `CMS_COLORS` or `CMS_X_COLORS` array. Page 37 of the reference manual incorrectly states that you can use `xv_get` to do this. XView ignores `CMS_COLOR_COUNT` with `xv_get`.

The first paragraph on page 37 (which describes `CMS_COLOR_COUNT`) should read:

"Used to specify the number of colors being set with `CMS_COLORS` or `CMS_X_COLORS`."

Supplementary XView Documentation

This section contains information regarding the XView toolkit that would normally reside in the O'Reilly documentation, but was not included in O'Reilly's final XView documentation release.

Joining Canvas Views

To join XView canvas views, destroy the view that you do not want by invoking `xv_destroy_safe`. Here is sample code to do this:

```
void
destroy_last_view(canvas)
Canvas canvas;
{
    /* destroy the last View window */
    Xv_Window view;
    int nviews;
    nviews = (int) xv_get(canvas, OPENWIN_NVIEWS);
    if (nviews > 1) {
        view = (Xv_Window) xv_get(canvas, OPENWIN_NTH_VIEW,
                                  nviews-1)
        (void) xv_destroy_safe(view);}
}
```

See the XView Programming Manual for information on how to split canvas views using `xv_set`.

XV_HELP_DATA

Set XV_HELP_DATA on a text subwindow by setting it on the text subwindow's view. You can obtain the view as follows:

```
xv_get(textsw, OPENWIN_NTH_VIEW, 0);
```

XV_FOCUS_RANK

XV_FOCUS_RANK specifies the focus client class for XView objects. It takes on values:

```
typedef enum
{
    XV_FOCUS_SECONDARY = 0, /* default value: Ordinary Focus */
    XV_FOCUS_PRIMARY = 1 /* First Class Focus */
} Xv_focus_rank;
```

The OPEN LOOK Mouseless Specification states that by default, textfields, numeric textfields, and scrolling lists are primary focus clients. Other XView controls default to secondary focus. The Specification also states that an application can choose to specify other controls as primary focus clients if necessary.

Scrollbars

This section discusses three scrollbar attributes: `SCROLLBAR_COMPUTE_SCROLL_PROC`, `SCROLLBAR_NORMALIZE_PROC`, and `SCROLLBAR_MOTION`.

See "Managing Your Own Scrollbar," in the XView Programming Manual's Scrollbar chapter, for more information on scrollbars.

SCROLLBAR_COMPUTE_SCROLL_PROC

This attribute has an associated function that converts physical scrollbar information into client object information. An example function call looks like:

```
scrollbar_compute_scroll_proc(sb, pos, available_cable, motion,
                    &offset, &object_length)
```

This function should return `offset` and `object_length`.

Use `default_compute_scroll_proc` to perform the normal scrollbar package functionality. If you do not set a normalize_proc, the offset becomes the viewstart (after bounds checking). The scrollbar package then scrolls the object to this offset.

Code Example 2-1 contains sample code for the `scrollbar_compute_scroll_proc` attribute's function.

Code Example 2-1 `scrollbar_compute_scroll_proc` attribute function

```
void
scrollbar_compute_scroll_proc(scrollpub, pos, avail_cable,
                    motion, offset, object_len)
    Scrollbar scrollpub;
```

Code Example 2-1 `scrollbar_compute_scroll_proc` attribute function

```
    int pos;
    int avail_cable;
    Scroll_motion motion;
    unsigned long *offset;
    unsigned long *object_len;
{
int new_start = TEXTSW_CANNOT_SET;
int lines = 0;

*obj_length = es_get_length(folio->views->esh);

switch(motion) {
case SCROLLBAR_ABSOLUTE:
    if (length == 0)
        new_start = pos;
    else
        new_start = *obj_length * pos / length;
    break;
case SCROLLBAR_POINT_TO_MIN:
case SCROLLBAR_MIN_TO_POINT: {
    if (lines == 0)
        lines++    /* Always make some progress */
    if (motion == SCROLLBAR_MIN_TO_POINT)
        lines = -lines;
    }
    break;
case SCROLLBAR_PAGE_FORWARD:
    lines = line_table.last_plus_one - 2;
    break;
case SCROLLBAR_PAGE_BACKWARD:
    lines = last_plus_one+ 2;
    break;
case SCROLLBAR_LINE_FORWARD:
    lines = 1;
    break;
case SCROLLBAR_LINE_BACKWARD:
    lines = -1;
    break;

case SCROLLBAR_TO_START:
    new_start = 0;
    break;
case SCROLLBAR_TO_END:
    new_start = *obj_length;
```

Code Example 2-1 `scrollbar_compute_scroll_proc` attribute function

```
    break;
default:
    break;
    }

xv_set(sb, SCROLLBAR_VIEW_LENGTH, last_plus_one - first, 0);
*offset = first;
return (XV_OK);
}
```

SCROLLBAR_NORMALIZE_PROC

This attribute's function takes the offset that the `scrollbar_compute_scroll_proc` attribute's function returns, and adjusts it. The scrollbar package then scrolls the object to this offset.

An example function call looks like:

```
scrollbar_normalize_proc(sb, voffset, motion, &vstart)
```

This function should return `vstart`. Code Example 2-2 contains sample code for the `scrollbar_normalize_proc` attribute's function.

Code Example 2-2 `scrollbar_normalize_proc` attribute function

```
scrollbar_normalize_proc(sb, offset, motion, vs)
    Scrollbar      sb;
    long unsigned  offset;
    Scroll_motion  motion;
    long unsigned  *vs; /* new offset == new viewstart */
{
line_ht = (int) xv_get(sb, SCROLLBAR_PIXELS_PER_UNIT);

/* If everything in the panel is in view, then don't scroll. */
if ((int) xv_get(sb, SCROLLBAR_OBJECT_LENGTH) <=
    (int) xv_get(sb, SCROLLBAR_VIEW_LENGTH))
    return (*vs = offset);

switch(motion){
case SCROLLBAR_ABSOLUTE:
case SCROLLBAR_LINE_FORWARD:
case SCROLLBAR_TO_START:
    align_to_max = TRUE;
```

Code Example 2-2 `scrollbar_normalize_proc` attribute function

```
    scrolling_up = TRUE;
    break;

case SCROLLBAR_PAGE_FORWARD:
case SCROLLBAR_TO_END:
    align_to_max = TRUE;
    scrolling_up = TRUE;
    break;

case SCROLLBAR_POINT_TO_MIN:
    align_to_max = TRUE;
    scrolling_up = TRUE;
    break;

case SCROLLBAR_MIN_TO_POINT:
    align_to_max = TRUE;
    scrolling_up = TRUE;
    break;

case SCROLLBAR_PAGE_BACKWARD:
case SCROLLBAR_LINE_BACKWARDS:
    align_to_max = FALSE;
    scrolling_up = FALSE;
    break; }
*vs = offset;
return(XV_OK);
}
```

SCROLLBAR_MOTION

The SCROLLBAR_MOTION attribute provides the scrolling motion resulting from a `scrollbar_request` event. The `xv_get` call:

```
xv_get(sb, SCROLLBAR_MOTION)
```

returns `motion`, which is one of:

- ABSOLUTE
- POINT_TO_MIN (from here_to_top on menu)
- PAGE_FORWARD
- LINE_FORWARD
- MIN_TO_POINT (from top_to_here on menu)

- PAGE_BACKWARD
- LINE_BACKWARD
- TO_END
- TO_START
- PAGE_ALIGNED

XView Panel Architecture

The XView Panel class architecture, inherited from SunView, does not sufficiently deal with container classes (numeric text fields, scrolling lists, and sliders) in a backward-compatible manner. This section discusses changes that address this issue.

XV_KEY_DATA *and* PANEL_CLIENT_DATA *Attributes*

Clients use `xv_set` along with `XV_KEY_DATA` or `PANEL_CLIENT_DATA` and an object handle to associate a key with a piece of data for the object. When an event occurs, the client can retrieve an object handle from the event proc. The client can then use the object handle (and `XV_KEY_DATA` or `PANEL_CLIENT_DATA`) to `xv_get` the data.

For numeric text fields, when an event occurs inside the text field, `PANEL_EVENT_PROC` is called with the handle to the text field as an argument. If the client retrieves data as described above, `xv_get` uses the text field's handle. This can cause problems, because the event is a numeric text field event. To eliminate any problems arising from this situation, whenever clients `xv_set` data for the numeric text field, it is automatically `xv_set` (using the same key and data) for the text field. Then `xv_get` will always retrieve the data associated with the numeric text field.

PANEL_ITEM_OWNER *Attribute*

The `PANEL_ITEM_OWNER` attribute is now public, so the client can obtain the parent container object from the child. If `PANEL_ITEM_OWNER` returns a non-`NULL` value, the client has obtained the handle to the parent object.

For example, for numeric text fields, `PANEL_ITEM_OWNER` called with the associated text field as an argument returns the numeric text field. `PANEL_ITEM_OWNER` returns `NULL` when called using the numeric text field as an argument.

XV_FOCUS_RANK *Attribute*

The XV_FOCUS_RANK attribute has been in the public header files since the XView 3.0 release. XV_FOCUS_RANK can be used with xv_set or xv_get. XV_FOCUS_RANK should only be used on panel items that accept keyboard input in full mouseless mode. For example, any panel item affected by the mouse or the keyboard: buttons, sliders, text items, etc.

XV_FOCUS_RANK can have one of these two values: XV_FOCUS_PRIMARY or XV_FOCUS_SECONDARY. Panel items with an XV_FOCUS_RANK or XV_FOCUS_PRIMARY accept keyboard input without full mouseless mode, such as panel text items and scrolling lists. Panel items with an XV_FOCUS_RANK of XV_FOCUS_SECONDARY only accept keyboard input if full mosueless mode is turned on, such as buttons and check boxes. If keyboard focus is moved into a panel, the upper leftmost XV_FOCUS_PRIMARY panel item is given the keyboard focus.

Using the Child Handle

Retrieve and set values using the child handle through the parent container object's API. Otherwise, your applications will not be forward-compatible beyond this release. Although you can obtain the child's handle, its API is private, so is subject to change.

Panel Drop Targets

The panel drop target has a new attribute, PANEL_DROP_DELETE. The default for PANEL_DROP_DELETE is TRUE. If you set it to FALSE, all drag and drop operations dropped on the PANEL_DROP_TARGET_ITEM behave as if the user presses the Control key (resulting in a copy operation). The attribute can be set in the panel notify procedure that is called when a drop occurs. This allows you to decide, while each drop is happening, whether to allow a drag move or only a drag copy. This is a boolean attribute, and can be used with xv_create(), xv_get(), and xv_set().

Compiling XView Programs

Use the following command line to compile your XView application:

```
cc -I$OPENWINHOME/include file.c -L$OPENWINHOME/lib -lxview
    -lolgx -lX11
```

You must specify the following libraries when statically linking an XView application:

- `-lxview`
- `-lolgx`
- `-lX11`
- `-lXext`
- `-lX11 (this is needed again by libXext)`
- `-lsocket`
- `-lnsl`
- `-lintl`
- `-lw`
- `-Bdynamic -ldl`

You must always link the `dl` library in dynamically. For example:

```
cc -I$OPENWINHOME/include foo.c -o foo -L$OPENWINHOME/lib
    -Bstatic -lxview -lolgx -lX11 -lXext -lX11 -lnsl -lintl -lw
    -Bdynamic -ldl
```

2

Part 2 — Internationalizing X View Applications

Introducing Internationalized XView

3

XView is a user-interface toolkit based on the X Window System and the OPEN LOOK graphical user interface. It is included in the current Solaris release.

XView allows you to internationalize your OPEN LOOK applications by supporting Asian and many western European languages:

- Western European (ISO Latin-1) languages and Asian languages are supported with locale settings, localized text handling, and customized object layout.
- Asian languages have multibyte, wide characters, input method, and font set support.

You do not have to redesign or recompile internationalized applications. The goal is to have a single application binary operate in any of the supported locales.

In an internationalized XView application, language-specific application data (message strings, labels, and so on) is separate from the rest of the application. To localize the application (that is, to adapt the application to support a specific language) you need to modify only the language-specific data. Thus, the task of porting an internationalized application consists of, among other things, translating application-specific strings and modifying object layout.

The information in this manual builds on the contents of several other manuals published by SunSoft and O'Reilly and Associates. Check the preface to make sure you have the manuals you need.

3

Internationalization Features

The internationalization of XView applications is outlined in the *XView Programming Manual, Version 3*, published by O'Reilly and Associates. Specifically, XView supports the following internationalization features described in that programming manual:

- *Locale setting*
  Before running an internationalized application, users must select what language to run in. The locale setting feature allows the user to choose the language or cultural environment.
- *Localized text handling*
  As a developer, you need to be able to write application strings—error messages, menu labels, button labels—in the native language, and have those strings retrieved in the language specified by the locale. This process is called localized text handling.
- *Object layout*
  When an application is run in a non-native language, the layout of various objects may change. For example, the dimensions of objects containing strings, such as buttons and panels, may change. Object layout is the mechanism by which the screen location of objects is modified (depending on the display language) to accommodate these kinds of changes.

This manual describes additional internationalization features that are *not* documented in the *XView Programming Manual.*

- Wide character and multibyte characters
- Input method
- Font sets

Wide Characters and Multibyte Characters

English language applications use ASCII encoding to represent characters. Each character is encoded using one byte (actually 7 out of the 8 bits). Other languages have multiple character sets that sometimes contain extremely large numbers of characters. These languages require more than one byte to represent each character and must be encoded differently. The current release of XView uses Extended UNIX Code (EUC) encoding.

Certain XView attributes and functions have been modified to handle EUC multibyte characters. There are also wide character attributes and functions. Wide character attributes are suffixed with `_WC` (wide character) or `_WCS` (wide character string). Similarly, wide character functions are suffixed with `_wc` or `_wcs`. See Chapter 4, “Character Encoding for more information.

Input Method

Input method refers to how users enter text in an application. For example, to enter data in a typical European language application—say German—users simply type the information. Many Asian languages, however, consist of multiple character sets (for example, Japanese has two phonetic alphabets and one ideographic character set). These multiple character sets can consist of many thousands of characters and contain numerous homonyms for any particular word. Entering data in these languages requires special input handling. See Chapter 5, “Input Method for more information.

Font Sets

Most western European languages consist of a single character set, and only one font is necessary to support the language. Languages with multiple character sets require multiple fonts, which are grouped into font sets. The font handling API has been extended in the current XView release to handle font sets. See “Fonts” on page 45 for more information.

Compiling XView Programs

Use the following command line to compile your XView application:

```
cc -DOW_I18N -I$OPENWINHOME/include file.c
-L$OPENWINHOME/lib -lxview -lolgx -lX11 {-lintl -lw}
```

`-DOW_I18N`= Enables internationalization support

`-lxview`= XView library

`-lolgx`= OPEN LOOK graphics library

`-lX11`= X11 Release 5 library

The following optional flags link in libraries that may be needed by the application:

`-lintl`=Message cataloguing library. Needed if application uses `gettext` family of functions.

`-lw=` Wide character support library. Needed if the application uses wide character functions such as `getwchar()`.

Character Encoding 4

In order to support a wide range of languages, The current XView release uses Extended UNIX Code (EUC) as its primary encoding method. EUC encoding is suited for internationalized applications because it is compatible with ASCII[1] and, at the same time, supports multiple character sets.

The character sets you use depend on the locale(s) associated with your application:

- *Non-Asian locales* usually have a single character set. For example, ASCII or ISO Latin-1 is suited for English or western European languages.
- *Asian locales* usually have multiple character sets.

EUC characters and text strings use either *multibyte* or *wide character* representation. In multibyte representation, characters are represented by a varying number of bytes. In wide character representation, characters are represented by a fixed number of bytes.

In the current XView release, attributes and functions have been modified to handle multibyte strings, and additional attributes and functions accommodate wide characters.

XView also uses Compound Text encoding for transferring data between X clients.

1. The multibyte API is compatible with earlier versions of XView, such as domestic U.S. XView 3.1, which used ASCII or ISO Latin-1 characters.

For detailed discussions on encoding, refer to these documents:

- EUC, multibyte, and wide character: *Developer's Guide to Internationalization.*
- Compound Text: *Compound Text Encoding, Version 1.1, MIT X Consortium Standard, X Version 11, Release 5* by Robert W. Scheifler.

Encodings Used in Asian Locales

As you write your program, you will need to choose a suitable character encoding and API. Figure 4-1 shows how you can use different encodings (EUC wide character and multibyte, and Compound Text) within the same application.

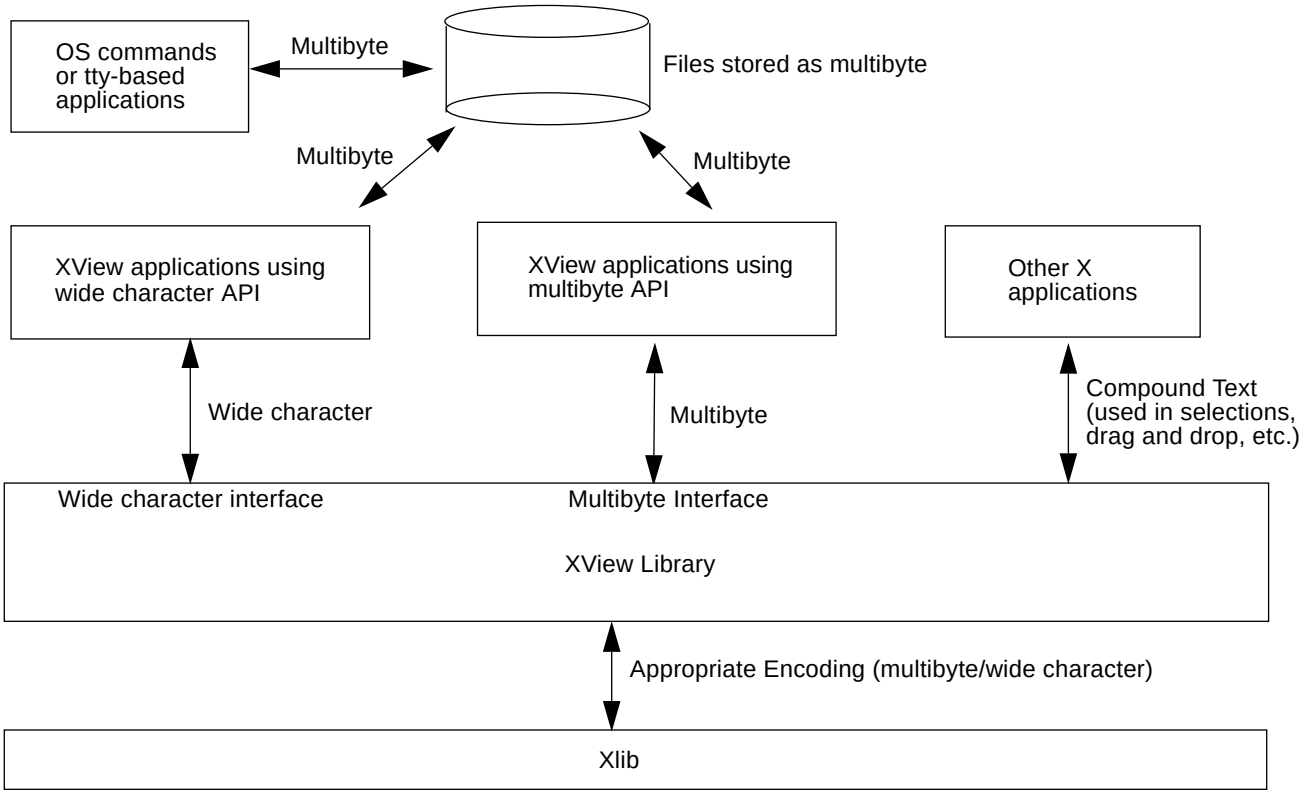


Figure 4-1 Encodings Used for Asian Locales

When to Use Multibyte and Wide Character

The wide character API (type `wchar_t`) consists of wide character string handling attributes and functions. The multibyte API (type `char`) is the same as in earlier, single-byte versions of XView.

You can mix wide character and multibyte characters within the same application. You can also mix wide character and multibyte APIs.

The XView library dynamically adjusts its internal data representation depending upon both the locale the application is running in and the nature of the data. As a result of this, programming convenience is the primary consideration regarding the choice of API.

When to Use Compound Text

The character encodings or character sets used in multibyte and wide character implementations may differ among vendors. For an application to communicate with other applications, a common encoding scheme is needed. XView relies on Compound Text, which is specified by the X Consortium.

Use Compound Text encoding in these situations:

- When an application needs to send a string composed of characters other than ISO Latin-1 across the X server to another application. XView uses Compound Text for data transfers to and from other X clients; for example, selection services (including drag and drop operations) and sending properties to the window manager
- When an application implements its own interclient communication; for example, a canvas-based application that uses selection service.

Note than an application is free to use a private encoding scheme for its own use, as long as the ICCCM is followed.

EUC Programming Issues

The following sections discuss special programming issues related to screen column definitions and passing multibyte strings.

Screen Columns

A screen column is defined as the pixel space required by a single ASCII character.[1] Asian characters may use a wider screen space than ASCII characters and are generally represented by more than one byte. Thus, in Asian locales:

screen columns != character count != byte count

Asian characters may also be interspersed with ASCII characters. In Asian locales, a fixed unit in pixels is needed to specify the space required by a screen column. Then wide Asian characters can occupy two or more columns, as in Figure 4-2.

Figure 4-2 ASCII and Japanese Characters

A number of functions and attributes use screen columns as arguments or returned values.

For example, `PANEL_VALUE_STORED_LENGTH` limits the number of characters that can be entered into a panel item. In Asian locales, `PANEL_VALUE_STORED_LENGTH` is measured in bytes. However, this attribute is screen-column based. If `PANEL_VALUE_STORED_LENGTH` and `PANEL_VALUE_DISPLAY_LENGTH` are specified to be 80, the user has allocated 80 screen columns, but not necessarily 80 characters, for display. In traditional Chinese, a Han character can be composed of 4 bytes and occupy 2 columns. Therefore, the `PANEL_VALUE_STORED_LENGTH` limit can be reached at 20 characters, yet only 40 screen columns are occupied.[2]

1. In previous releases (such as domestic U.S. XView 3.1), a screen column was defined as the space occupied by one character, and one character was represented by one byte. Therefore, the following used to apply:

```
screen columns==character count==byte count
```

Wide Character Attributes and Functions

Most XView multibyte attributes and functions that take a string or character as an argument have wide character analogs. These wide character attributes and functions have similar names composed of the original names suffixed with `_WCS`, `_WC`, `_wc`, or `_wcs`:

- `_WC` for wide character attributes
- `_wc` for wide character functions
- `_WCS` for wide character string attributes
- `_wcs` for wide character string functions

2. The screen column concept is only applicable in the case of fixed-width fonts. By default, the C locale uses fixed-width fonts for `textsw` and `ttysw`, and variable-width fonts for frame and panel. Currently, Asian locales use only fixed-width fonts.

4

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.

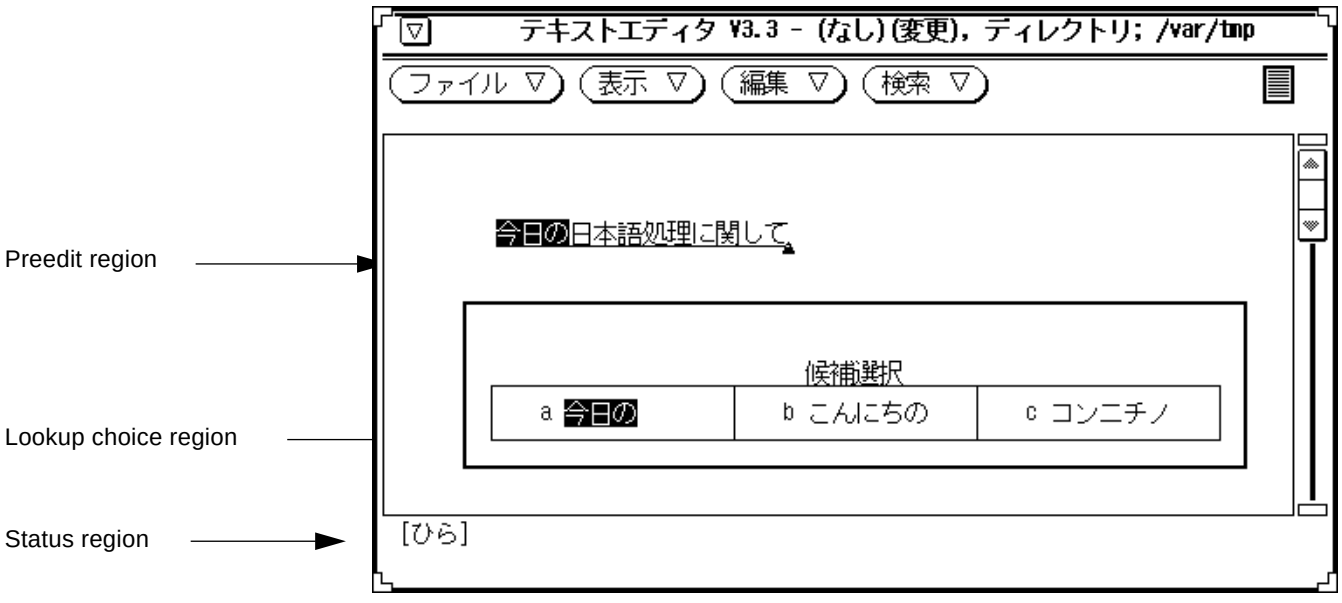


Figure 5-1 Japanese Input Method Screen Regions

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.

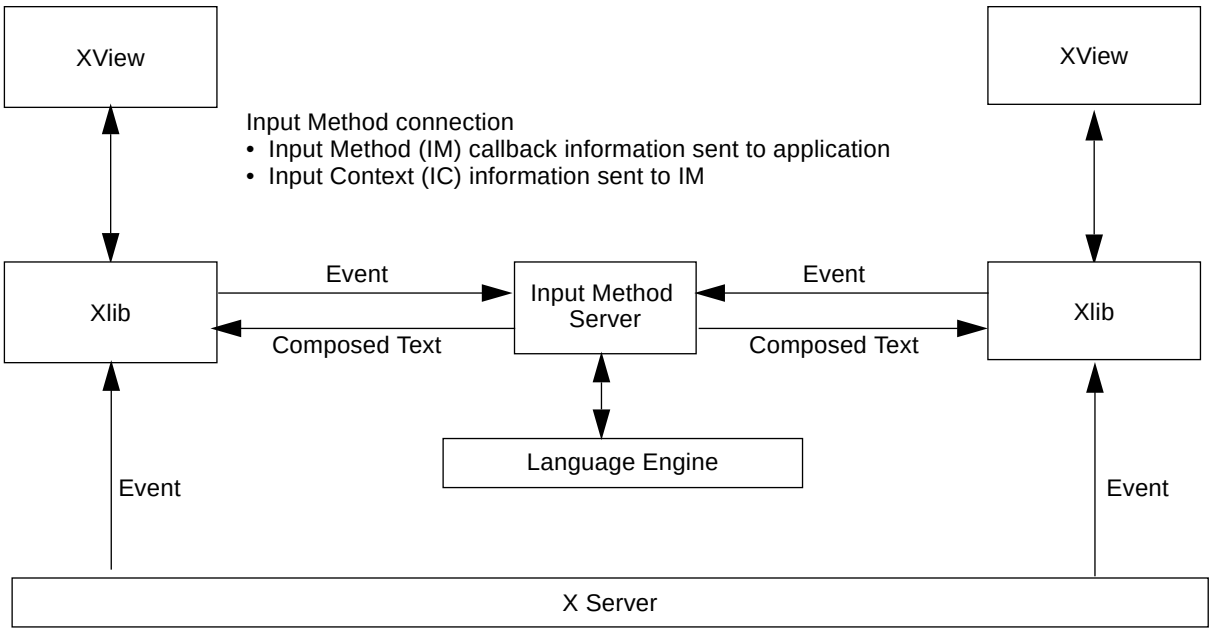


Figure 5-2 High-level Overview of Input Method

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

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 x`view.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

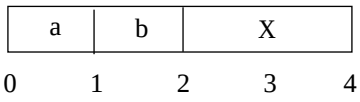
Wide Character Attributes	Wide Character Function
TEXTSW_ACTION_CHANGED_DIRECTORY_WCS	textsw_append_file_name_wcs()
TEXTSW_ACTION_EDITED_FILE_WCS	textsw_store_file_wcs()
TEXTSW_ACTION_LOADED_FILE_WCS	
TEXTSW_FILE_WCS	
TEXTSW_FILE_CONTENTS_WCS	
TEXTSW_INSERT_FROM_FILE_WCS	

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

API Summaries

This appendix is divided into two sections: attributes and functions. Each section describes how to use the APIs. Use the information in this appendix as an addendum to the *XView Reference Manual* by O'Reilly & Associates.

Attributes

This section lists attributes alphabetically and summarizes each attribute's purpose, type, default, and procs.

CANVAS_IM_PREEDIT_FRAME

Provides a pointer to the preedit popup window's frame.

Type: `Frame`
Default: none
Procs: `get`

Only the following attributes can be set with predictable results for the frame object returned by `CANVAS_IM_PREEDIT_FRAME`: `XV_LABEL`, `XV_SHOW`, `XV_X`, `XV_Y`, `XV_WIDTH`, `XV_HEIGHT`.

CURSOR_STRING_WCS

Creates a text drag and drop cursor. If the wide character string exceeds three characters (not bytes), then only the first 3 characters are displayed, and "the more arrow" is shown inside the cursor.

Type: `wchar_t *`
Default: none
Procs: `create`, `get`

FILE_CHOOSER_APP_DIR_WCS

Adds an application-specific pathname to the Go To history menu's fixed space.

Type: `wchar_t *` pair
Default: none
Procs: `create`, `set`

FILE_CHOOSER_CUSTOMIZE_OPEN_WCS

Allows the client to use the Open window in certain contexts: Insert, Choose, Import.

Type: `wchar_t *, wchar_t *, enum`
Default: none
Procs: `create`

FILE_CHOOSER_DIRECTORY_WCS

Specifies the current working directory that is displayed in Open, Save, and Save As windows.

Type: `wchar_t *`
Default: Current working directory name
Procs: `create`, `get`, `set`

FILE_CHOOSER_DOC_NAME_WCS

In a Save window, specifies the default document name ("Untitled1").
In a Save As window, specifies the current document name.

Type: `wchar_t *`
Default: Untitled1 (in a Save window)
Procs: `create`, `get`, `set`

FILE_CHOOSER_FILTER_STRING_WCS

Sets or gets an ex-like regular expression string. The string's files are filtered before being displayed in the list. If not filter string is specified, all entries are assumed to match. "`..`" is always assumed to match.

Type: `wchar_t *`
Default: none
Procs: `create`, `get`, `set`

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

FILE_CHOOSER_NOTIFY_FUNC_WCS

Invokes a callback when the user selects a file from an Open, Save, or Save As window. Expected return values are `XV_OK` or `XV_ERROR`. The form of the callback depends on whether the `FILE_CHOOSER_TYPE` attribute is set to `FILE_CHOOSER_OPEN`, `FILE_CHOOSER_SAVE`, or `FILE_CHOOSER_SAVE_AS`.

Type: Function pointer
Default: NULL
Procs: `create`, `get`, `set`

FILE_CHOOSER_WCHAR_NOTIFY

Specifies that the callbacks registered by `FILE_LIST_CHANGE_DIR_FUNC`, `FILE_LIST_FILTER_FUNC`, and `FILE_LIST_COMPARE_FUNC` receive wide character strings, not character strings.

Type: `Boolean`
Default: FALSE
Procs: `create`, `get`, `set`

FILE_LIST_DIRECTORY_WCS

Sets or gets the directory currently displayed in the list.
If `FILE_LIST_DIRECTORY` is set to `NULL`, the directory will be empty.

Type: `wchar_t *`
Default: Name of the current working directory
Procs: `create`, `get`, `set`

FILE_LIST_DOTDOT_STRING_WCS

Allows the client to modify the string used by the file list package to denote the "`..`" entry.

Type: `wchar_t *`
Default: "`...Go up one folder...`"
Procs: `create`, `get`, `set`

FILE_LIST_FILTER_STRING_WCS

Sets or gets an ex-like regular expression string. The string's files are filtered before being displayed in the list. If not filter string is specified, all entries are assumed to match. "`..`" is always assumed to match.

Type: `wchar_t *`
Default: none
Procs: `create`, `get`, `set`

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

FILE_LIST_WCHAR_NOTIFY

Specifies that the callbacks registered by `FILE_LIST_CHANGE_DIR_FUNC`, `FILE_LIST_FILTER_FUNC`, and `FILE_LIST_COMPARE_FUNC` receive wide character strings, not character strings.

Type: `Boolean`
Default: FALSE
Procs: `create`, `get`, `set`

FONT_CHAR_HEIGHT_WC

Returns the height (`int`) of a specified wide character (`wchar_t`).

Type: `int`
Default: none
Procs: `get`

FONT_CHAR_WIDTH_WC

Returns the width (`int`) of a specified wide character (`wchar_t`).

Type: `int`
Default: none
Procs: `get`

FONT_COLUMN_WIDTH

Returns the default screen column width in pixels, given a specified XView font set object.

Type: `int`
Default: none
Procs: `get`

FONT_LOCALE

Specifies the locale for a font set object.

Type: `char *`
Default the basic locale
Procs: `create`, `find`

FONT_NAMES

Specifies the XLFD font names of the fonts to be used to construct the font set object.

Type: `char **`
Default: none
Procs: `create`, `find`, `get`

FONT_SET_ID

Returns the X font set, when given an XView font set object.

Type: `XFontSet`
Defaults: none
Procs: `get`

FONT_SET_SPECIFIER

Specifies the name of a font set. The font set definition database will be queried with the specified value.

Type: `char *`
Default: none
Procs: `create`, `find`, `get`

FONT_STRING_DIMS_WCS

Given a wide character string and the address of a `Font_string_dims` structure, `xv_get()` fills in and returns a pointer to the structure describing the pixel dimensions of the string.

Type: `Font_string_dims *`
Default: none
Procs: `get`

FRAME_LABEL_WCS

Specifies the label used in the window manager's titlebar for the frame. This is the wide character version of `FRAME_LABEL`.

Type: `wchar_t *`
Default: `NULL`
Procs: `create`, `get`, `set`

FRAME_LEFT_FOOTER_WCS

Specifies the left-justified footer. This is the wide character version of `FRAME_LEFT_FOOTER`.

Type: `wchar_t *`
Default: `NULL`
Procs: `create`, `get`, `set`

FRAME_RIGHT_FOOTER_WCS

Specifies the right-justified footer. This is the wide character version of `FRAME_RIGHT_FOOTER`.

Type: `wchar_t *`
Default: `NULL`
Procs: `create`, `get`, `set`

HISTORY_ADD_FIXED_ENTRY_WCS

Adds a string to the fixed space in the list. A fixed string is added to the bottom of the fixed space in the list. Passing a label of NULL adds a blank menu item.

Type: wchar_t * pair
Default: none
Procs: create, set

HISTORY_ADD_ROLLING_ENTRY_WCS

Adds a string to the rolling space. Strings in the rolling space and stacked and "roll off" after a specified number of strings are added.

Type: wchar_t *
Default: none
Procs: create, set

HISTORY_LABEL_WCS

Returns the label from a specified space (defined as HISTORY_FIXED or HISTORY_ROLLING) for a specified row number. If the row does not exist, NULL is returned.

Type: Int pair
Default: none
Procs: get

HISTORY_NOTIFY_PROC_WCS

Invokes a callback-based on the user's selection from the Go To history menu.

Type: function pointer
Default: none
Procs: create, set

HISTORY_VALUE_WCS

Returns the VALUE from a specified space (defined as HISTORY_FIXED or HISTORY_ROLLING) for a specified row number. If the row does not exist, NULL is returned.

Type: Int pair
Default: none
Procs: get

ICON_LABEL_WCS

Specifies the icon label used for the frame. This is the wide character string version of ICON_LABEL.

Type: wchar_t *
Default: NULL
Procs: create, get, set

ICON_TRANSPARENT_LABEL_WCS

Draws the given wide character string into an icon. It does not affect any other pixels in the bounding box. This is the wide character version of ICON_TRANSPARENT_LABEL.

Type: wchar_t *
Default: NULL
Procs: create, get, set

MENU_ACCELERATOR_WCS

Sets an accelerator on a menu item when used in a create or set call. If an accelerator is changed with xv_set(), FRAME_MENUS must be set again before the accelerator takes effect. XView copies the wide character accelerator string, and get returns the accelerator string. Do not modify the return string.

Type: wchar_t *
Default: none
Procs: create, get, set

MENU_ACTION_ACCELERATOR_WCS

Creates a menu item with a given label, notify proc, and accelerator. If an accelerator is changed with xv_set(), FRAME_MENUS must be set again before the accelerator takes effect. XView copies the wide character accelerator string, but not the wide character label string.

Type: `wchar_t *, void (*)(), wchar_t *`
Default: none
Procs: `create, set`

MENU_ACTION_ITEM_WCS

Provides a shortcut for creating or modifying a menu item and associating it with a notify procedure. It takes two values: a wide character string and a notify procedure. This is the wide character version of `MENU_ACTION_ITEM`.

Type: `wchar_t *, (void *)()`
Default: Not applicable
Procs: `create, set`

MENU_GEN_PIN_WINDOW_WCS

Uses a command window to create the pin window for a menu. The contents are based on the static (not dynamic) menu contents. The frame is the parent frame; the name in wide characters is the name of the pin window. All menu items must have a notify procedure; `MENU_NOTIFY_PROC` for the menu itself is ignored. This is the wide character version of `MENU_GEN_PIN_WINDOW`.

Type: `Frame, wchar_t *`
Default: no pin window
Procs: `create, set`

MENU_GEN_PROC_ITEM_WCS

Defines the generate procedure and wide character text string for a menu item. This is the wide character version of `MENU_GEN_PROC_ITEM`.

Type: `wchar_t *, (* Menu)()`
Default: `NULL`
Procs: `create, set`

MENU_GEN_PULLRIGHT_ITEM_WCS

Provides a shortcut for creating a menu item (or a menu item's submenu) and associating it with a pull-right menu generate procedure. This is the wide character version of `MENU_GEN_PULLRIGHT_ITEM`.

Type: `wchar_t *, (* Menu) ()`
Default: none
Procs: `create, set`

MENU_PULLRIGHT_ITEM_WCS

Creates a string menu item with pull-right submenu. This is the wide character version of `MENU_PULLRIGHT_ITEM`.

Type: `wchar_t *, Menu`
Default: Not applicable
Procs: `create, set`

MENU_STRING_WCS

Sets or gets the string of a given menu item. This is the wide character version of `MENU_STRING`.

Type: `wchar_t *`
Default: `NULL`
Procs: `create, get, set`

MENU_STRING_ITEM_WCS

Defines the text string and value for a menu item. This is the wide character version of `MENU_STRING_ITEM`.

Type: `wchar_t *, Xv_opaque`
Default: `NULL`
Procs: `create, set`

MENU_STRINGS_WCS

Sets the strings for multiple menu items. This is the wide character version of `MENU_STRINGS`.

Type: list of `wchar_t *`
Default: Not applicable
Procs: `create, set`

MENU_STRINGS_AND_ACCELERATORS_WCS

Creates menu items with the given labels and accelerators. If an accelerator is changed with `xv_set()`, `FRAME_MENUS` must be set again before the accelerator takes effect. XView copies the wide character accelerator strings, but not the wide character label strings.

Type: list of `<label (wchar_t *), accelerator (wchar_t *)> pairs, terminated by NULL`
Default: none
Procs: `create`, `set`

MENU_TITLE_ITEM_WCS

Set the title string of a menu. Can only be used with menus that do not originate from pull-right items or pulldown menu buttons. This is the wide character version of `MENU_TITLE_ITEM`.

Type: `wchar_t *`
Default: No title
Procs: `create`, `set`

NOTICE_BUTTON_WCS

Specifies a wide character string to be displayed in a button and a value to use if the button is selected. This is the wide character version of `NOTICE_BUTTON`.

Type: `wchar_t *`, `int`
Default: none
Procs: `create`, `set`

NOTICE_BUTTON_NO_WCS

Specifies a wide character string associated with the NO (cancel) button of a notice. The value returned if this button is selected is `NOTICE_NO`. This is the wide character version of `NOTICE_BUTTON_NO`.

Type: `wchar_t *`
Default: none
Procs: `create`, `set`

NOTICE_BUTTON_YES_WCS

Specifies a wide character string associated with the YES (confirm) button. The value returned if this button is selected is NOTICE_YES. This is the wide character version of NOTICE_BUTTON_YES.

Type: wchar_t *
Default: none
Procs: create, set

NOTICE_MESSAGE_STRING_WCS

Specifies the text to print in a notice. The value of this attribute is a wide character string. This is the wide character version of NOTICE_MESSAGE_STRING.

Type: wchar_t *
Default: NULL
Procs: create, set

NOTICE_MESSAGE_STRINGS_WCS

Specifies the text to print in a notice. The value of this attribute is a NULL- terminated list of wide character strings. This is the wide character version of NOTICE_MESSAGE_STRINGS.

Type: list of wchar_t *
Default: NULL
Procs: create, set

NOTICE_MESSAGE_STRINGS_ARRAY_PTR_WCS

Specifies the text to print in a notice. The value of this attribute is a variable pointing to a NULL-terminated array of wide character strings. This is the wide character version of NOTICE_MESSAGE_STRINGS_ARRAY_PTR.

Type: wchar_t **
Default: NULL
Procs: create, set

PANEL_CHOICE_STRING_WCS

Sets the wide character string for a choice item. The first argument to the attribute is the index of a choice item, the second argument is the wide character string. This is the wide character version of PANEL_CHOICE_STRING.

Type: int, wchar_t *
Default: NULL
Procs: create, get, set
Objects: Panel_choice_item

PANEL_CHOICE_STRINGS_WCS

The value of this attribute is a NULL terminated list of wide character strings. Each string will be the label of a choice. This is the wide character version of PANEL_CHOICE_STRINGS.

Type: list of wchar_t *
Default: NULL
Procs: create, get, set
Objects: Panel_choice_item

PANEL_ITEM_IC_ACTIVE

Specifies whether a panel text item should allow or disallow input method conversion. The default value is set according to the value of the WIN_USE_IM attribute of the parent window.

Type: Boolean
Default: TRUE
Procs: create, get, set
Objects: Panel_text_item, Panel_slider_item, Panel_list_item, Panel_numeric_text_item

PANEL_LABEL_STRING_WCS

Specifies the wide character string for an item's label. This is the wide character version of PANEL_LABEL_STRING.

Type: wchar_t *
Default: NULL
Procs: create, get, set

PANEL_LIST_INSERT_STRINGS_WCS

Inserts the specified wide character strings into the scrolling list before the specified row. The first argument to the attribute is a row number, and the second argument is a pointer to a NULL-terminated array of wide character strings. This is the wide character version of PANEL_LIST_INSERT_STRINGS.

Type: int, wchar_t **
Default: Not applicable
Procs: create, set
Objects: Panel_list_item

PANEL_LIST_ROW_VALUES_WCS

Gets row values from or sets row values in PANEL_LIST. Takes the row number, a pointer to a Panel_list_row_values_wcs structure, 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_image   glyph;
    Server_image   mask_glyph;
    Xv_font    font
    Xv_opaque  client_data;
    Xv_opaque  extension_data;
    unsigned   inactive : 1;
    unsigned   selected : 1;
} Panel_list_row_values_wcs;
```

Type: int, struct pointer, int
Default: none
Procs: create, get, set

PANEL_LIST_STRING_WCS

Assign the specified wide character string (second argument) to the specified row (third argument). xv_get() will return a pointer to the string in the specified row. This is the wide character version of PANEL_LIST_STRING.

Type: int, wchar_t *
Default: NULL
Procs: create, get, set
Objects: Panel_list_item

PANEL_LIST_STRINGS_WCS

Similar to its companion attribute, PANEL_LIST_STRING, except that it takes a NULL-terminated list of wide character strings as its value. This is the wide character version of PANEL_LIST_STRINGS.

Type: list of wchar_t *
Return type: wchar_t *
Default: NULL
Procs: create, get, set
Objects: Panel_list_item

PANEL_LIST_TITLE_WCS

The title of a scrolling list as a wide character string. This is the wide character version of PANEL_LIST_TITLE.

Type: wchar_t *
Default: NULL
Procs: create, get, set
Objects: Panel_list_item

PANEL_MASK_CHAR_WC

When the user enters a character, the character specified as the value of PANEL_MASK_CHAR_WC will be displayed in place of each character the user has typed. Use the space character for no character echo (caret does not advance). Use the NULL character to disable masking. If PANEL_MASK_CHAR_WC or PANEL_MASK_CHAR is specified for a particular panel text item, then preedit input will be disabled for this panel text item. This is the wide character version of PANEL_MASK_CHAR.

Type: wchar_t
Default: none
Procs: create, get, set
Objects: Panel_text_item, Panel_numeric_text_item

PANEL_MAX_TICK_STRING_WCS

Specifies the wide character string which appears underneath the maximum tick mark on horizontal sliders, or to the right of the maximum tick mark on vertical sliders. PANEL_MAX_TICK_STRING_WCS is ignored if PANEL_TICKS is 0. The width of the slider can be adjusted to insure that there is enough space to accommodate both the minimum and maximum tick strings.

Type: wchar_t *
Default: NULL
Procs: create, get, set
Objects: Panel_slider_item, Panel_gauge_item

PANEL_MAX_VALUE_STRING_WCS

Maximum value string in wide characters for the slider. On horizontal sliders, the maximum value string appears to the right of the maximum end box. On vertical sliders, the maximum value string appears above the maximum end box. This is the wide character version of PANEL_MAX_VALUE_STRING.

Type: wchar_t *
Default: NULL
Procs: create, get, set
Objects: Panel_slider_item

PANEL_MIN_TICK_STRING_WCS

Wide character string that appears underneath the minimum tick mark on vertical sliders. PANEL_MIN_TICK_STRING_WCS is ignored if PANEL_TICKS is 0. The width of the slider can be adjusted to insure that there is enough room to accommodate both the minimum and maximum tick strings. This is the wide character version of PANEL_MIN_TICK_STRING.

Type: wchar_t *
Default: NULL
Procs: create, get, set
Objects: Panel_slider_item, Panel_gauge_item

PANEL_MIN_VALUE_STRING_WCS

Minimum value wide character string for the slider. On horizontal sliders, the minimum value string appears to the left of the minimum end box. On vertical sliders, the minimum value string appears below the minimum end box. This is the wide character version of PANEL_MIN_VALUE_STRING.

Type: wchar_t *
Default: NULL
Procs: create, get, set
Objects: Panel_slider_item

PANEL_NOTIFY_PROC_WCS

Procedure to call when a scrolling list is activated. This is the wide character version of PANEL_NOTIFY_PROC.

Argument: See below
Default: none
Procs: create, get, set
Objects: Panel_item

Callback:

```
int
notify_proc_name(list_item, wide_char_string, client_data, op,
        event, row)
    Panel_item list_item;     /* PANEL_LIST item */
    wchar_t    *wide_char_string; /* string associated with row */
    Xv_opaque  client_data;   /* Client data of the row*/
    Panel_list_op   op;       /* Select, validate or
             * delete operation */
    Event   *event;
    int     row;       /* Scroll list row number*/
```

PANEL_NOTIFY_STRING_WCS

The value is a wide character string that triggers notification when typed into a text item. Applies only when PANEL_NOTIFY_LEVEL is PANEL_SPECIFIED. This is the wide character version of PANEL_NOTIFY_STRING.

Type: wchar_t *
Default: \r\t\n (carriage return, tab, newline)
Procs: create, get, set
Objects: Panel_multiline_text_item,Panel_numeric_text_item, Panel_text_item

PANEL_VALUE_WCS

Indicates the current value of a panel item as a wide character string if the value is a string. This is the wide character version of PANEL_VALUE.

Type: wchar_t *
Default: NULL
Procs: create, get, set
Objects: Panel_choice_item, Panel_gauge_item, Panel_list_item,Panel_multiline_text_item, Panel_numeric_text_item Panel_slider_item, Panel_text_item

PANEL_VALUE_DISPLAY_LENGTH

Maximum number of columns to display in a text item. The length of the value display cannot be less than the combined width of the left and right “more text” buttons. Note that this definition is different from the XView 3.1 environment.

Type: int
Default: 80
Procs: create, get, set
Objects: Panel_list_item, Panel_slider_item, Panel_numeric_text_item, Panel_text_item

PANEL_VALUE_STORED_LENGTH

Maximum number of bytes to be stored in the value string for text items. The panel value of a panel text item will be converted to multibyte to check whether it has the reached the limit set by this attribute.

Type: int
Default: 80
Procs: create, get, set
Objects: Panel_list_item,

```
Panel_multiline_text_item,
Panel_numeric_text_item,
Panel_text_item
```

PANEL_VALUE_STORED_LENGTH_WCS

Maximum number of wide characters to be stored in the value string for text items.

Type: `int`
Default: 80
Procs: `create, get, set`
Objects:
```
Panel_list_item,
Panel_multiline_text_item,
Panel_numeric_text_item,
Panel_text_item
```

PATH_LAST_VALIDATED_WCS

Returns the last pathname that has passed validation in the text field. If no pathname has passed validation, `NULL` is returned. The return value is set to `NULL` after setting `PATH_IS_DIRECTORY`.

`PATH_LAST_VALIDATED_WCS` returns the expanded version of the pathname, in contrast to `PANEL_VALUE`, which returns the current contents of the field.

Type: `wchar_t *`
Default: none
Procs: `get`

PATH_RELATIVE_TO_WCS

Specifies an absolute path to which any relative path input is appended to complete the pathname.

Type: `wchar_t *`
Default: none
Procs: `create`, `get`, `set`

SELN_REQ_CHARSIZE

Specifies the number of characters in the selection.

Type: `int`
Default: none
Procs: `selection_ask()`, `selection_init_request()`, `selection_query()`

SELN_REQ_CONTENTS_WCS

Specifies a pointer to a `wchar_t` string containing the selection's wide character contents.

Type: `wchar_t *`
Default: none
Procs: `selection_ask()`, `selection_init_request()`, `selection_query()`

SELN_REQ_FIRST_WC

Gives the number of characters that precede the first character of the selection.

Type: `int`
Default: none
Procs: `selection_ask()`, `selection_init_request()`, `selection_query()`

SELN_REQ_LAST_WC

Gives the character index of the last character of the selection.

Type: `int`
Default: none
Procs: `selection_ask()`, `selection_init_request()`, `selection_query()`

SERVER_IMAGE_BITMAP_FILE_WCS

Specifies a filename in wide characters for the file containing the X11 bitmap to be created. Refer to the Xlib documentation on `XReadBitmapFile()` for the file format. This is the wide character version of `SERVER_IMAGE_BITMAP_FILE`.

Type: `wchar_t *`
Default: none
Procs: `create`

TEXTSW_ACTION_CHANGED_DIRECTORY_WCS

The current working directory for the process has changed to the directory named. The name of the directory is specified in wide characters. This is the wide character version of TEXTSW_ACTION_CHANGED_DIRECTORY.

Type: wchar_t *
Default: none
Procs: notify_proc

TEXTSW_ACTION_EDITED_FILE_WCS

The file named by the provided wide character string has been edited. This is the wide character version of TEXTSW_ACTION_EDITED_FILE.

Type: wchar_t *
Default: none
Procs: notify_proc

TEXTSW_ACTION_LOADED_FILE_WCS

The text subwindow is used to view the file named. The filename is specified with a wide character string by the provided wide character string.

Type: wchar_t *
Default: none
Procs: notify_proc

TEXTSW_CONTENTS_WCS

Specifies the text for a text subwindow as a wide character string. xv_get() needs additional parameters. This is the wide character version of TEXTSW_CONTENTS.

Type: wchar_t *
Default: NULL
Procs: create, get, set

The following example shows how to use TEXTSW_CONTENTS_WCS to read text from a text subwindow.

```
wchar_t    *buf
int     buf_len;   /* character base */
Textsw_index   start_pos, next_pos;  /* character base */

next_pos = xv_get(textsw, TEXTSW_CONTENTS_WCS,
                                    start_pos, buf, buf_len);
```

TEXTSW_FILE_WCS

For xv_create() and xv_set(), specifies the name of the file to load in wide character format. For xv_get(), returns the name of the file loaded in wide character or NULL, if no file was loaded.

Type: wchar_t *
Default: NULL
Procs: create, get, set

TEXTSW_FILE_CONTENTS_WCS

Initializes the text subwindow contents from a file, yet still edits the contents in memory as if specified using TEXTSW_FILE_CONTENTS_WCS. The filename is specified in wide character. This is the wide character version of TEXTSW_FILE_CONTENTS.

Type: wchar_t *
Default: NULL
Procs: create, set

TEXTSW_FIRST_WC

Specifies the zero-based index in characters of the first displayed character. This is the wide character version of TEXTSW_FIRST.

Type: int
Default: none
Procs: get

TEXTSW_INSERT_FROM_FILE_WCS

Inserts the contents of a file, whose name is given as a wide character string, into a textsw at the current insertion point.

Type: `wchar_t *`
Default: `NULL`
Procs: `create`, `set`

TEXTSW_INSERTION_POINT_WC

Specifies the character-based index of the current insertion point. This is the wide character version of `TEXTSW_INSERTION_POINT`.

Type: `Textsw_index`
Default: none
Procs: `create`, `get`, `set`

TEXTSW_LENGTH_WC

`TEXTSW_LENGTH_WC` returns the length in characters of the `textsw`'s contents. This is dissimilar to `TEXTSW_LENGTH`, which returns the length in bytes.

Type: int
Default: none
Procs: `get`

WIN_ERROR_MSG_WCS

Specifies a wide character error message string. This is the wide character version of `WIN_ERROR_MSG`.

Type: `wchar_t *`
Default: `NULL`
Procs: `create`

WIN_IC

Sets or gets the input context (IC) handle associated with a given window. This will only affect windows which have `WIN_USE_IM` set to `TRUE`.

Type: `int`
Default: none
Procs: `create`, `get`, `set`

WIN_IC_ACTIVE

Specifies whether the input context (IC) associated with the given window is to be active. If WIN_IC_ACTIVE is TRUE, events will be directed to the input method. The attribute should be set to TRUE whenever the user expects to use the input method. WIN_IC_ACTIVE can also be set to FALSE when it is necessary to temporarily disable the input method (IM) for a particular window.

Type: Boolean
Default: TRUE
Procs: create, set, get

WIN_IC_COMMIT_STRING

If the window has an input context (IC) associated with it, querying this attribute will return the currently committed string for this window. The committed string is any input string that may have been pending before WIN_IC_RESET was set to TRUE.

Type: char *
Default: NULL
Procs: get

WIN_IC_COMMIT_STRING_WCS

If the window has an input context (IC) associated with it, querying this attribute will return the currently committed string for this window. The committed string is any input string that may have been pending before WIN_IC_RESET was set to TRUE. This is the wide character version of WIN_IC_COMMIT_STRING.

Type: wchar_t *
Default: NULL
Procs: get

WIN_IC_CONVERSION

Specifies whether input method conversion mode is on for a given window. This will only affect windows which have WIN_USE_IM set to TRUE.

Type: Boolean
Default: FALSE
Procs: create, get, set

WIN_IC_PREEDIT_CARET

Specifies the preedit caret callback and the associated callback data for a given window. The specified function is called when the input method wants the client to move the text insertion caret. This will only affect windows which have WIN_USE_IM set to TRUE.

Type: void (*preedit_start_proc)(), XPointer
Default: window dependent
Procs: create, set

WIN_IC_PREEDIT_DONE

Specifies the preedit done callback and the associated callback data for a given window. The specified function is called when input method conversion is turned off. This will only affect windows which have WIN_USE_IM set to TRUE.

Type: void (*preedit_done_proc)(), XPointer
Default: window dependent
Procs: create, set

WIN_IC_PREEDIT_DRAW

Specifies the preedit draw callback and the associated callback data for a given window. The specified function is called when the input method wants the client to draw, insert, delete, or replace preedit text in the preedit region. This will only affect windows which have WIN_USE_IM set to TRUE.

Type: void (*preedit_draw_proc)(), XPointer
Default: window dependent
Procs: create, set

WIN_IC_PREEDIT_START

Specifies the preedit start callback and the associated callback data for a given window. The specified function is called when input method conversion is turned on. This will only affect windows which have WIN_USE_IM set to TRUE.

Type: `void (*preedit_start_proc)()`, XPointer
Default: window dependent
Procs: `create`, `set`

WIN_IC_RESET

Resets the input context of a given window to its initial state. Any input pending on that input context is deleted. The input method clears the preedit area and updates the status area accordingly. If `WIN_IC_RESET` is set to `TRUE`, the input method conversion mode is turned off.

Type: `Boolean`
Default: none
Procs: `set`

WIN_IC_STATUS_DONE

Specifies the status done callback function and associated callback data for a given window. The status area is an output-only region which is intended to present the internal state of the input method to the user. The specified function is called when the input context loses focus or is destroyed. This will only affect windows which have `WIN_USE_IM` set to `TRUE`.

Type: `void (*status_done_proc)()`, XPointer
Defaults: private default function
Procs: `create`, `set`

WIN_IC_STATUS_DRAW

Specifies the status draw callback function and associated callback data for a given window. The status area is an output-only region which is intended to present the internal state of the input method to the user. The specified function is called when the input method wants the client to update the status area. This will only affect windows which have `WIN_USE_IM` set to `TRUE`.

Type: `void (*status_draw_proc)()`, XPointer
Defaults: private default function
Procs: `create`, `set`

WIN_IC_STATUS_START

Specifies the status start callback function and associated callback data for a given window. The status area is an output-only region which is intended to present the internal state of the input method to the user. The specified function is called when the input context is created or receives focus. This will only affect windows which have WIN_USE_IM set to TRUE.

Type: void (*status_start_proc)(), XPointer
Defaults: private default function
Procs: create, set

WIN_USE_IM

Specifies whether the input method will be enabled (TRUE) or disabled (FALSE) for a given window. WIN_USE_IM can be set on a frame, canvas, panel, text subwindow, or tty subwindow. The value of WIN_USE_IM can only be set with xv_create(). If WIN_USE_IM is not explicitly set, subwindows will inherit their value of WIN_USE_IM from the parent.

Type: Boolean
Default: TRUE
Procs: create, get

WIN_X_IM_STYLE_MASK

Specifies the input style mask for the input method, which will be specified as either preedit or status style. The mask is specified using XIMStyle masks in <X11/Xlib.h>. The value can be queried against any window object.

If WIN_USE_IM is TRUE, this attribute affects the input context associated with a window. XView attribute settings override input style command line options and input style resource settings.

Type: unsigned long
Default: locale-specific
Procs: xv_create(), xv_get()

XV_IM

Returns the XIM handle of the Xlib connection with the input method. XV_IM returns NULL if a connection to the input method has not been established. The server package attempts to make a connection during xv_init() only if the locale-specific resource xview.needIM.<locale> is TRUE. By default, xview.needIM.<locale> is FALSE.

The xview.needIM.<locale> resource is typically specified as TRUE in $OPENWINHOME/lib/locale/<locale>/xview/defaults, if the locale supports an input method. For the C locale and most western European locales, an input method may not be required; therefore, the default value of FALSE will suffice.

Type: XIM
Default: NULL
Procs: get

XV_IM_STYLES

Returns the XIMStyles list of supported styles. This attributes can be queried on any window or server object.

Type: XIMStyles
Default: none
Procs: xv_get()

XV_LABEL_WCS

Specifies a frame's header label or an icon's label, or simply associates a wide character name with an object. XView copies the strings on set. This is the wide character version of XV_LABEL.

Type: wchar_t *
Default: NULL
Procs: create, get, set

Functions

This section lists functions alphabetically.

defaults_get_locale()

Returns the current locale setting in the XView resource database lookup, as specified by the first argument for `defaults_set_locale()`.

```
char *
defaults_get_locale()
```

defaults_set_locale()

Turns the locale-sensitive XView resource database lookup on and off. If both `<locale>` and `<locale_attr>` are `NULL`, a call to this function turns off the locale-sensitive resource lookup (default). If `<locale>` or `<locale_attr>` is non-`NULL`, the locale-sensitive XView resource database lookup is turned on. `<locale>` must be a valid locale name, and `<locale_attr>` must be `XV_LC_BASIC_LOCALE`, `XV_LC_DISPLAY_LANG`, `XV_LC_NUMERIC`, or `XV_LC_TIME_FORMAT`.

```
void
defaults_set_locale(locale, locale_attr)
    char   *locale;          /* Locale value itself */
   int     locale_attr;      /* Locale category */
```

panel_get_value_wcs()

Macro that returns the `PANEL_VALUE_WCS` for the specified panel item. This is the wide character version of `panel_set_value()`.

```
wchar_t *
panel_get_value_wcs(item)
  Panel_item     item;
```

panel_set_value_wcs()

Macro that sets `PANEL_VALUE_WCS` for the specified panel item. This is the wide character version of `panel_get_value()`.

```
panel_set_value_wcs (item, value)
  Panel_item     item;
  wchar_t *      value;
```

textsw_add_mark_wc()

Adds a new mark at a character-based position. Flags can be either TEXTSW_MARK_DEFAULTS or TEXTSW_MARK_MOVE_AT_INSERT. This is the wide character version of textsw_add_mark().

```
Textsw_mark
textsw_add_mark_wc(textsw, position, flags)
          Textsw          textsw;
          Textsw_index    position;  /* character base */
          unsigned        flags;
```

textsw_append_file_name_wcs()

Returns 0 if textsw is editing a file and appends the name of the file at the end of name. This is the wide character version of textsw_append_file_name().

```
int
textsw_append_file_name_wcs(textsw, name)
Textsw          textsw;
wchar_t         *name;
```

textsw_delete_wcs()

Removes the span of characters beginning with first and ending one before last_plus_one. Returns 0 if the operation fails; otherwise, it returns the number of characters (not bytes) deleted. This is the wide character version of textsw_delete().

```
Textsw_index  /* character base */
textsw_delete_wcs (textsw, first, last_plus_one)
          Textsw         textsw;
          Textsw_index   first, last_plus_one; /*character base*/
```

textsw_edit_wcs()

Erases a character, a word, or a line, depending on whether unit is TEXTSW_UNIT_IS_CHAR, TEXTSW_UNIT_IS_WORD, or TEXTSW_UNIT_IS_LINE. If direction is 0, characters after the current insertion point are affected; otherwise, characters before the current insertion point are erased. The operation is repeated count times. The function returns 0 on failure. This is the wide character version of textsw_edit().

```
Textsw_index    /* character base */
textsw_edit_wcs (textsw, unit, count, direction)
          Textsw         textsw;
          unsigned       unit, count, direction;
```

textsw_erase_wcs()

Equivalent to `textsw_delete_wcs()`, but does not affect the global shelf. Returns 0 if the operation fails, otherwise, returns the number of characters (not bytes) actually deleted. This is the wide character version of `textsw_erase()`.

```
Textsw_index  /* character base */
textsw_erase_wcs (textsw, first, last_plus_one)
          Textsw         textsw;
          Textsw_index   first, last_plus_one; /* character base*/
```

textsw_find_wcs()

Beginning at the position addressed by `first`, searches for the pattern specified by `buf` of length `buf_len`. Searches forward if flag is 0; otherwise, it searches backward. Returns -1 if there is no match; otherwise, the matching span is placed in indices addressed by `first` and `last_plus_one`. This is the wide character version of `textsw_find()`.

```
int
textsw_find_wcs(textsw, first, last_plus_one, buf,
                buf_len, flags)
          Textsw         textsw;
          Textsw_index   *first, *last_plus_one;/*char. base*/
          wchar_t        *buf;
          unsigned       buf_len; /* character base */
          unsigned       flags;
```

textsw_find_mark_wc()

Returns the current position of mark in characters. If this operation fails, the function returns `TEXTSW_INFINITY`. This is the wide character version of `textsw_find_mark()`.

```
Textsw_index             /* character base */
textsw_find_mark_wc(textsw, position, flags)
          Textsw         textsw;
          Textsw_mark    position;
```

textsw_index_for_file_line_wc()

Returns the character index of the first character in the line given by `line`. If this operation fails, the function returns `TEXTSW_CANNOT_SET`. This is the wide character version of `textsw_index_for_file_line()`.

```
Textsw_index            /* character base */
textsw_index_for_file_line_wc(textsw, line)
          Textsw        textsw;
          int           line;
```

textsw_insert_wcs()

Insert wide characters from `buf` into `textsw` at the current insertion point. The number of characters (not bytes) actually inserted is returned. This will equal `buf_len` unless there was memory allocation failure. If there was a failure, the return value is 0. This is the wide character version of `textsw_insert()`.

```
Textsw_index            /* character base */
textsw_insert_wcs (textsw, buf, buf_len)
          Textsw        textsw;
          wchar_t       *buf;
          int           buf_len; /* character base */
```

textsw_match_wcs()

Searches for a block of text in the contents of a textsw. This is the wide character version of `textsw_match_bytes()`.

```
int
textsw_match_wcs(textsw, first, last_plus_one, start_sym
            start_sym_len, end_sym, end_sym_len, field_flag)

          Textsw       textsw;
          Textsw_index *first, *last_plus_one;/*character base*/
          wchar_t      *start_sym, *end_sym;
          int          start_sym_len, end_sym_len; /*char.base*/
          unsigned     field_flag;
```

textsw_normalize_view_wc()

Repositions the text so that the character at `position` (character base) is visible at the top of the subwindow. This is the wide character version of `textsw_normalize_view()`.

```
void
textsw_normalize_view_wc(textsw, position)
          Textsw         textsw;
          Textsw_index   position; /* character base */
```

textsw_possibly_normalize_wc()

If the character at the character-based `position` is already visible, this function does nothing. If the character is not visible, this function repositions the text so that the character is visible and at the top of the subwindow. This is the wide character version of `textsw_possibly_normalize()`.

```
void
textsw_possibly_normalize_wc(textsw, position)
          Textsw         textsw;
          Textsw_index   position;/* character base */
```

textsw_replace_wcs()

Replaces the character span from first to `last_plus_one` with the characters in `buf`. The return value is the number of characters inserted or deleted. The number is positive if characters are inserted, negative if characters are deleted. (The number is also negative if the original string is longer than the one that replaces it.) If this operation fails, it returns a value of `NULL`.

```
Textsw_index            /* character base */
textsw_replace_wcs(textsw, first, last_plus_one,
                           buf, buf_len)
          Textsw          textsw;
          Textsw_index    first, last_plus_one; /*char. base */
          wchar_t        *buf;
          unsigned        buf_len; /* character base */
```

textsw_set_selection_wcs()

Sets the selection to begin at `first` and include all characters up to `last_plus_one`. A `type` value of 1 indicates primary selection; 2 indicates secondary selection. This is the wide character version of `textsw_set_selection()`.

```
void
textsw_set_selection_wcs(textsw,first,last_plus_one, type)
          Textsw        textsw;
          Textsw_index  first, last_plus_one; /*character base */
          unsigned      type;
```

textsw_store_file_wcs()

Stores the contents of `textsw` to the file named by `filename`. If needed, a message box is displayed at x, y. This is the wide character version of `textsw_store_file()`.

```
unsigned
textsw_store_file_wcs (textsw, filename, x, y)
          Textsw          textsw;
          wchar_t        *filename;
          int             x, y;
```

ttysw_input_wcs()

Appends `len` number of wide characters from `buf` into input queue of the tty. The function returns the number of wide characters accepted. This is the wide character version of `ttysw_input()`.

```
int
ttysw_input_wcs (tty, buf, len)
          Tty           tty;
          wchar_t       *buf;
          int           len;
```

ttysw_output_wcs()

Appends `len` number of wide characters from `buf` into output queue of the tty. Characters are sent through the terminal emulator to the tty. It returns the number of wide characters accepted. This is the wide character version of `ttysw_output_wcs()`.

```
          int
          ttysw_output_wcs (tty, buf, len);
          Tty           tty;
          wchar_t       *buf;
          int           len;
```

Changes to Internationalized XView Version 2.x

B

Prior to the release of XView 3.2, XView was available in domestic and internationalized versions. This appendix explains the changes that have been made since XView 2.x. It is intended for developers who are migrating their internationalized XView 2.x applications to the current XView release. Topics include compatibility, changes to packages, and a list of current attributes and functions.

Compatibility with the Current XView Release

Programs written under the internationalized XView 2.x environment are neither source nor binary compatible with the current XView release. Programs written under the internationalized XView Version 2.x environment may require source modification to become compatible with XView. Table B-1 shows compatibility between toolkit versions.

Table B-1 Source Compatibility Matrix

Written/Compiled	**Operating Environment**		
	current XView release	**XView 3.0.1 and 3.1**	**Intl. XView 2.0.1**
current XView release	n/a	Not compatible	Not compatible
XView 3.0.1 and 3.1	Source/binary compatible	n/a	Not compatible
Intl. XView 2.0.1	Not compatible	Not compatible	n/a

B

Package Changes

This section describes changes to four packages: frames, panels, text subwindows, and windows.

Frames

Frame attributes have not changed since internationalized XView 2.x. In the internationalized XView 2.x releases, the frame package sent input method status information to the window manager, olwm, and then the window manager rendered the status region. In the current XView release, the frame package maintains the input method status information and renders the status region.

Panels

Three attributes in panels have changed meaning since internationalized XView 2.x:

- `PANEL_VALUE_DISPLAY_LENGTH`
- `PANEL_VALUE_STORED_LENGTH`
- `PANEL_NOTIFY_PROC_WCS`.

In internationalized XView 2.x and (domestic) XView 3.x, `PANEL_VALUE_DISPLAY_LENGTH` measured the display length in characters. However, the meanings of characters, bytes, and columns are not the same in the Asian locales; therefore, the meaning of `PANEL_VALUE_DISPLAY_LENGTH` also had to change to reflect the difference. In the current XView release, `PANEL_VALUE_DISPLAY_LENGTH` is measured in columns. The default value for `PANEL_VALUE_DISPLAY_LENGTH` is 80 columns.

`PANEL_VALUE_STORED_LENGTH` was also measured in characters in internationalized XView 2.x. Setting `PANEL_VALUE_STORED_LENGTH` to 10 meant 10 bytes of storage space in XView 2.x, and possibly 20 bytes of storage in internationalized XView 2.x (depending on the definition of a wide character). To make things more consistent in all environments, the value associated with `PANEL_VALUE_STORED_LENGTH` now means the number of bytes to be stored. All incoming wide characters are converted to multibyte first to check against the storage limit, then stored in wide character form. All incoming multibyte characters are also checked against the storage limit first, then converted to wide characters for storage.

PANEL_NOTIFY_PROC_WCS accepted five parameters for a panel list item in internationalized XView 2.x. In the XView release, PANEL_NOTIFY_PROC_WCS accepts six parameters for a panel list item:

- Panel list item
- Wide character string
- Client data
- Operation
- Event
- Row number.

Text Subwindows

The following sections describe changes to Text subwindows.

Search Path for Textsw Extras Menu File

The search path for the Textsw Extras Menu file has been changed since XView 2.x. Refer to “Text Subwindows” on page 63 for further information.

Character- and Byte-Based Index Support

In internationalized XView 2.x there was no API for a byte-based index. Indices were handled in characters, even for the multibyte API. In the current XView release, Text subwindows support both character- and byte-based APIs for index. The multibyte API handles the index in bytes, and the wide character API handles it in characters. There are two reasons for this change:

- To provide portability between Asian and non-Asian locales.
- To enhance the ease of programming with the multibyte API.

The attributes and functions in Table B-2 use an index as an argument or the returned value, or return the number of deleted/inserted characters.

- Column A lists functions and attributes that existed in the internationalized XView 2.x. Indices and deleted or inserted length were handled in characters, not bytes. In the current XView release, these functions and attributes are handled in bytes.

- Column B lists functions and attributes that existed in the internationalized XView 2.x as wide character API, where indices and deleted lengths were handled in characters. In the current XView release, these API handle indices and lengths in bytes similar to the multibyte API.
- Column C lists the new wide character API. This API supports wide characters as the counterpart of the multibyte API in column B. Indices and deleted length are handled in characters.

Table B-2 Changed or Added Text Subwindow Attributes and Functions

Multibyte API		Wide Character API
A	B	C
TEXTSW_CONTENTS	TEXTSW_FIRST	TEXTSW_FIRST_WC
	TEXTSW_INSERTION_POINT	TEXTSW_INSERTION_POINT_WC
textsw_find_bytes()	textsw_add_mark()	textsw_add_mark_wc()
textsw_insert()	textsw_delete()	textsw_delete_wcs()
textsw_match_bytes()	textsw_edit()	textsw_edit_wcs()
textsw_replace_bytes()	textsw_erase()	textsw_erase_wcs()
	textsw_find_mark()	textsw_find_mark_wc()
	textsw_index_for_file_line()	textsw_index_for_file_line_wc()
	textsw_normalize_view()	textsw_normalize_view_wc()
	textsw_possibly_normalize()	textsw_possibly_normalize_wc()
	textsw_set_selection()	textsw_set_selection_wcs()

If an internationalized XView 2.x application does not use any wide character API for the multibyte API in column A, or TEXTSW_LENGTH_WC (listed in Table B-3), the application will not need to change except for the changes according to the bug fixes mentioned in the next section. However, the wide character API is recommended for improved performance. Refer to “Text Subwindows” on page 63 for further information.

Table B-3 Wide Character Text Subwindow API

Wide Character Attributes	Wide Character Functions
TEXTSW_CONTENTS_WCS	textsw_find_wcs()
TEXTSW_LENGTH_WC	textsw_insert_wcs()
	textsw_match_wcs()
	textsw_replace_wcs()

If an internationalized XView 2.x application uses the wide character API listed in Table B-3 and any API listed in column B of Table B-2 together, the application needs to use the wide character API listed in column C instead of column B of Table B-2. For example, the internationalized XView 2.x code below searches a string, erases it, and moves the insertion caret to the front of the erased character.

```
Textsw textsw;
Textsw_index first, last_plus_one;

textsw_find_wcs (textsw, &first, &last_plus_one, buf, buf_len, flags);
textsw_erase (textsw, first, last_plus_one);
xv_set (textsw, TEXTSW_INSERTION_POINT, first, NULL);
```

The XView code below has been changed to call `textsw_erase_wcs()` instead of `textsw_erase()`, and it uses `TEXTSW_INSERTION_POINT_WC` instead of `TEXTSW_INSERTION_POINT`. `Textsw_find_wcs()` returns the index parameters in wide characters. Further processing of the indices should also be character-based by using `textsw_erase_wcs()` and `TEXTSW_INSERTION_POINT_WC`.

```
textsw_find_wcs (textsw, &first, &last_plus_one, buf, buf_len, flags);
textsw_erase_wcs (textsw, first, last_plus_one);
xv_set (textsw, TEXTSW_INSERTION_POINT_WC, first, NULL);
```

If an internationalized XView 2.x application uses the multibyte API in column A of Table B-2 and wide character API in Table B-3 together, some combinations will require source code changes. For example, the following internationalized XView 2.x code searches a string with a multibyte string and replaces it with a wide character string.

```
char            *buf;
wchar_t         *wbuf;

textsw_find_bytes (textsw, &first, &last_plus_one,buf, buf_len, flags);
textsw_replace_wcs (textsw, first, last_plus_one, wbuf, wbuf_len);
```

In the current XView release, `textsw_find_bytes()` should be replaced with `textsw_find_wcs()`, and the multibyte string `buf` should be converted to wide character. Similarly, `textsw_replace_wcs()` can be replaced with `textsw_replace_bytes()`, but the wide character API is recommended for better performance. These changes are necessary because the index is handled in different ways between multibyte and wide character APIs.

Bug Fixes to Text Subwindows

Because of these bug fixes, your internationalized XView Version 2.x application may need to change.

`textsw_insert()`

```
Textsw_index
textsw_insert (textsw, buf, buf_len)
  char          *buf;
  int           buf_len;
```

Internationalized XView Version 2.x environment: `buf_len` and the return value of the number of characters actually inserted were counted in characters. This was a bug. If `buf_len` is specified over the actual buffer length, only the contents of the buffer would be inserted.

Current XView release: `buf_len` and the return value are counted in bytes.

`textsw_replace_bytes()`

```
Textsw_index
textsw_replace_bytes(textsw, first, last_plus_one,
                buf, buf_len)
  Textsw_index first, last_plus_one;
  char          *buf;
  unsigned       buf_len;
```

Internationalized XView Version 2.x environment: `buf_len`, the return value of the number of characters inserted or deleted, `first`, and `last_plus_one` were measured in characters. Measuring `buf_len` in characters was a bug.

Current XView release: `buf_len`, return value, `first`, and `last_plus_one` are counted in bytes.

`textsw_find_bytes(), textsw_match_bytes()`

```
textsw_find_bytes (textsw, first, last_plus_one, buf,
  buf_len, flags)
```

```
textsw_match_bytes(textsw,first,last_plus_one, start_sym,
  start_sym_len, end_sym, end_sym_len, field_flag)
```

Internationalized XView Version 2.x environment: these functions ignore the specified buffer length and processes the entire string in buffer (this is a bug). The index is character based.

Current XView release: processes string in buffer specified by buffer length. Index and buffer length are byte based.

`TEXTSW_CONTENTS`

`TEXTSW_CONTENTS` is used by `xv_get()`.

next_pos = xv_get (textsw, TEXTSW_CONTENTS, start_pos, buf, buf_len)

```
Textsw_index start_pos, next_pos;
char       *buf;
int        buf_len;
```

Internationalized XView Version 2.x environment: `next_pos` and `start_pos` are character based. `buf_len` is byte based.

Current XView release: `start_pos`, `next_pos` and `buf_len` are byte based.

Windows

As shown in Table B-4, the `WIN_IM_*` attributes in internationalized XView 2.x have undergone a name change to `WIN_IC_*` in the current XView release. The API usage is still the same, although the Xlib data structures have been updated to reflect the latest X11 Release 5 specification.

Table B-4 WIN_IM_* Attribute Changes

Internationalized XView 2.x	Current XView Release
WIN_IM_PREEDIT_START	WIN_IC_PREEDIT_START
WIN_IM_PREEDIT_DRAW	WIN_IC_PREEDIT_DRAW
WIN_IM_PREEDIT_DONE	WIN_IC_PREEDIT_DONE
WIN_IM_LUC_START	No longer supported
WIN_IM_LUC_DRAW	No longer supported
WIN_IM_LUC_PROCESS	No longer supported
WIN_IM_LUC_DONE	No longer supported
WIN_IM_STATUS_START	WIN_IC_STATUS_START
WIN_IM_STATUS_DRAW	WIN_IC_STATUS_DRAW
WIN_IM_STATUS_DONE	WIN_IC_STATUS_DONE

XView Attributes and Functions

In Table B-5, current XView attributes and functions are listed alphabetically, and the status is designated as:

*New:*New since internationalized XView 2.x

*Modified:*Existed in internationalized XView 2.x but is modified in the current XView release

*Unchanged:*Existed in internationalized XView 2.x and remains unchanged in the current XView release

*Obsolete:*Existed in internationalized XView 2.x but is no longer used in the current XView release

Table B-5 XView Attributes and Functions—Current Release

Attributes and Functions	New	Modified	Unchanged	Obsolete
CANVAS_IM_PREEDIT_FRAME	x			
CURSOR_STRING_WCS	x			
defaults_get_locale()	x			
defaults_set_locale()	x			

Table B-5 XView Attributes and Functions—Current Release (continued)

Attributes and Functions	New	Modified	Unchanged	Obsolete
FILE_CHOOSER_APP_DIR_WCS	x			
FILE_CHOOSER_CUSTOMIZE_OPEN_WCS	x			
FILE_CHOOSER_DIRECTORY_WCS	x			
FILE_CHOOSER_DOC_NAME_WCS	x			
FILE_CHOOSER_FILTER_WCS	x			
FILE_CHOOSER_NOTIFY_FUNC_WCS	x			
FILE_CHOOSER_WCHAR_NOTIFY	x			
FILE_LIST_DIRECTORY	x			
FILE_LIST_DOTDOT_STRING_WCS	x			
FILE_LIST_FILTER_STRING_WCS	x			
FILE_LIST_WCHAR_NOTIFY	x			
FONT_CHAR_WIDTH_WC			x	
FONT_CHAR_HEIGHT_WC			x	
FONT_COLUMN_WIDTH	x			
FONT_LOCALE	x			
FONT_NAMES	x			
FONT_SET_ID	x			
FONT_SET_SPECIFIER	x			
FONT_STRING_DIMS_WCS	x			
FRAME_LABEL_WCS			x	
FRAME_LEFT_FOOTER_WCS			x	
FRAME_RIGHT_FOOTER_WCS			x	
HISTORY_ADD_FIXED_ENTRY_WCS	x			
HISTORY_ADD_ROLLING_ENTRY_WCS	x			
HISTORY_LABEL_WCS	x			
HISTORY_NOTIFY_PROC_WCS	x			
HISTORY_VALUE_WCS	x			
ICON_LABEL_WCS			x	
ICON_TRANSPARENT_LABEL_WCS			x	

Table B-5 XView Attributes and Functions—Current Release (continued)

Attributes and Functions	New	Modified	Unchanged	Obsolete
MENU_ACCELERATOR_WCS	x			
MENU_ACTION_ACCELERATOR_WCS	x			
MENU_ACTION_ITEM_WCS			x	
MENU_GEN_PIN_WINDOW_WCS			x	
MENU_GEN_PROC_ITEM_WCS			x	
MENU_GEN_PULLRIGHT_ITEM_WCS			x	
MENU_PULLRIGHT_ITEM_WCS			x	
MENU_STRING_WCS			x	
MENU_STRING_ITEM_WCS			x	
MENU_STRINGS_WCS			x	
MENU_STRINGS_AND_ACCELERATORS_WCS	x			
MENU_TITLE_ITEM_WCS			x	
NOTICE_BUTTON_WCS			x	
NOTICE_BUTTON_NO_WCS			x	
NOTICE_BUTTON_YES_WCS			x	
NOTICE_MESSAGE_STRING_WCS			x	
NOTICE_MESSAGE_STRINGS_WCS			x	
NOTICE_MESSAGE_STRINGS_ARRAY_PTR_WCS			x	
PANEL_CHOICE_STRING_WCS			x	
PANEL_CHOICE_STRINGS_WCS			x	
PANEL_ITEM_IC_ACTIVE	x			
PANEL_LABEL_STRING_WCS			x	
PANEL_LIST_INSERT_STRINGS_WCS	x			
PANEL_LIST_ROW_VALUES_WCS	x			
PANEL_LIST_STRING_WCS			x	
PANEL_LIST_STRINGS_WCS			x	
PANEL_LIST_TITLE_WCS	x			
PANEL_MASK_CHAR_WC	x			
PANEL_MAX_TICK_STRING_WCS	x			
PANEL_MAX_VALUE_STRING_WCS	x			
PANEL_MIN_TICK_STRING_WCS	x			
PANEL_MIN_VALUE_STRING_WCS	x			

Table B-5 XView Attributes and Functions—Current Release (continued)

Attributes and Functions	New	Modified	Unchanged	Obsolete
PANEL_NOTIFY_PROC_WCS		x		
PANEL_NOTIFY_STRING_WCS			x	
PANEL_VALUE_WCS			x	
PANEL_VALUE_DISPLAY_LENGTH		x		
PANEL_VALUE_STORED_LENGTH		x		
PANEL_VALUE_STORED_LENGTH_WCS	x			
panel_get_value_wcs()			x	
panel_set_value_wcs()			x	
PATH_LAST_VALIDATED_WCS	x			
PATH_RELATIVE_TO_WCS	x			
SELN_REQ_CHARSIZE			x	
SELN_REQ_CONTENTS_WCS			x	
SELN_REQ_FIRST_WC	x			
SELN_REQ_LAST_WC	x			
SERVER_IMAGE_BITMAP_FILE_WCS			x	
TEXTSW_ACTION_CHANGED_DIRECTORY_WCS			x	
TEXTSW_ACTION_EDITED_FILE_WCS			x	
TEXTSW_ACTION_LOADED_FILE_WCS			x	
TEXTSW_CONTENTS		x		
TEXTSW_CONTENTS_WCS			x	
TEXTSW_FILE_WCS			x	
TEXTSW_FILE_CONTENTS_WCS			x	
TEXTSW_FIRST		x		
TEXTSW_FIRST_WC	x			
TEXTSW_INSERT_FROM_FILE_WCS			x	
TEXTSW_INSERTION_POINT		x		
TEXTSW_INSERTION_POINT_WC	x			
TEXTSW_LENGTH_WC			x	
textsw_add_mark()		x		
textsw_add_mark_wc()	x			
textsw_append_file_name_wcs()			x	
textsw_delete()		x		
textsw_delete_wcs()	x			

Table B-5 XView Attributes and Functions—Current Release (continued)

Attributes and Functions	New	Modified	Unchanged	Obsolete
textsw_edit()		x		
textsw_edit_wcs()	x			
textsw_erase()		x		
textsw_erase_wcs()	x			
textsw_find_bytes()		x		
textsw_find_wcs()			x	
textsw_find_mark()		x		
textsw_find_mark_wc()	x			
textsw_index_for_file_line()		x		
textsw_index_for_file_line_wc()	x			
textsw_insert()		x		
textsw_insert_wcs()			x	
textsw_match_bytes()		x		
textsw_match_wcs()			x	
textsw_normalize_view()		x		
textsw_normalize_view_wc()	x			
textsw_possibly_normalize()		x		
textsw_possibly_normalize_wc()	x			
textsw_replace_bytes()		x		
textsw_replace_wcs()			x	
textsw_set_selection()		x		
textsw_set_selection_wcs()	x			
textsw_store_file_wcs()			x	
ttysw_input_wcs()	x			
ttysw_output_wcs()	x			
WIN_ERROR_MSG_WCS	x			
WIN_IC			x	
WIN_IC_ACTIVE	x			
WIN_IC_COMMMIT_STRING	x			
WIN_IC_COMMIT_STRING_WCS	x			
WIN_IC_CONVERSION	x			
WIN_IC_PREEDIT_CARET	x			
WIN_IC_PREEDIT_DONE			x	
WIN_IC_PREEDIT_DRAW			x	
WIN_IC_PREEDIT_START			x	

Table B-5 XView Attributes and Functions—Current Release (continued)

Attributes and Functions	New	Modified	Unchanged	Obsolete
WIN_IC_RESET	x			
WIN_IC_STATUS_DONE			x	
WIN_IC_STATUS_DRAW			x	
WIN_IC_STATUS_START			x	
WIN_IM_LUC_START				x
WIN_IM_LUC_DRAW				x
WIN_IM_LUC_PROCESS				x
WIN_IM_LUC_DONE				x
WIN_USE_IM			x	
WIN_X_IM_STYLE_MASK	x			
XV_IM			x	
XV_IM_STYLES	x			
XV_LABEL_WCS			x	

B

Font Set Definitions

This appendix describes the contents of the font set definition database:

- Font set specifier
- Font set name aliases
- Default font family
- Default font scales
- Font family, scales, and sizes aliases

Font Set Specifier

A font set is specified by using the font set name as the resource specification and the corresponding list of XLFD font names as the resource value. The following example shows the font set name “-sun-myoungjo-medium-r-normal--16-140-75-75-p-140-korean-0” has been defined to consist of two fonts. The keyword `definition` indicates that the value following it is a list of XLFD font names.

```
-sun-myoungjo-medium-r-normal--16-140-75-75-p-140-korean-0:definition,\
    -b&h-lucida-medium-r-normal-sans-0-0-0-0-p-0-iso8859-1, \
    -sun-myoungjo-medium-r-normal--16-140-75-75-c-140-ksc5601.1987-0
```

An application can specify the font set to be used by assigning the value “-sun-myoungjo-medium-r-normal--16-140-75-75-p-140-korean-0” to the attribute `FONT_SET_SPECIFIER`.

Font Set Name Aliases

Other font set names can be aliased to a particular font set name using the keyword `alias`. The following example aliases the name “hngmnj14” to the font set name “-sun-myoungjo-medium-r-normal--16-140-75-75-c-140-korean-0”.

```
hngmnj14: alias, -sun-myoungjo-medium-r-normal--16-140-75-75-c-140-korean-0
```

An application can also specify the font set to be used by assigning the value “hngmnj14” to the attribute `FONT_SET_SPECIFIER`.

Default Font Family

Font attributes `FONT_FAMILY` can be used to create or find a font set object. The default font family for a particular locale can be defined in the font set definition database file as below:

```
!       Default Font Set
xv_font_set.default_family: FONT_FAMILY_SANS_SERIF
```

The font family `FONT_FAMILY_SANS_SERIF` has been defined as the default for the `FONT_FAMILY` attribute. If the default font family is not specified in the font set definition database, the C locale default is used. Refer to the *XView Programming Manual, Version 3* for these C locale defaults.

Default Font Scales

The point sizes corresponding to the font scales of a particular locale can be specified in the font set definition database in the following manner:

```
xv_font_set.small: 12
xv_font_set.medium: 14
xv_font_set.large: 16
xv_font_set.extra_large : 20
```

The point size defined by `xv_font_set.medium` is used as the default value for the `FONT_SIZE` attribute. If the default font scales are not specified in the font set definition database, the C locale default is used. Refer to the *XView Programming Manual, Version 3* for these C locale defaults.

Font Family, Scales, and Size Aliases

When an application creates a font set using `FONT_FAMILY` in the Asian locales, the font set definition database will be queried using a concatenation of the font set family name, the style, and the sizes as the resource specification. Therefore, font set aliases must exist in the font set definition database corresponding to the default font families, default font scales, and sizes. For example:

```
FONT_FAMILY_SANS_SERIF-FONT_STYLE_NORMAL-12:alias, \
    -sun-gothic-medium-r-normal--14-120-75-75-c-120-korean-0
FONT_FAMILY_SANS_SERIF-FONT_STYLE_NORMAL-14:alias, \
    -sun-myoungjo-medium-r-normal--16-140-75-75-p-140-korean-0
FONT_FAMILY_SANS_SERIF-FONT_STYLE_NORMAL-16:alias, \
    -sun-gothic-medium-r-normal--18-160-75-75-c-160-korean-0
FONT_FAMILY_SANS_SERIF-FONT_STYLE_NORMAL-20:alias, \
    -sun-myoungjo-medium-r-normal--22-200-75-75-c-200-korean-0
FONT_FAMILY_SANS_SERIF-FONT_STYLE_NORMAL-24:alias, \
    -sun-myoungjo-medium-r-normal--26-240-75-75-c-240-korean-0
```

If an application does not specify the `FONT_FAMILY`, `FONT_STYLE` or `FONT_SIZE` explicitly, the default values of these attributes for the particular locale are used. In the following example, the font set “-sun-myoungjo-medium-r-normal--16-140-75-75-p-140-korean-0” would be created.

```
Xv_Font font_set;
font_set = xv_find(frame, FONT,
        FONT_LOCALE, “ko”,
        NULL);
```

C

Part 3 — Release

XView Release Notes D

This appendix contains information pertinent to the current XView release.

notify.h Header File

The XView public header file `notify.h` no longer includes the header file <`sys/rusage.h`>. You might have a compilation problem if your code references <`sys/rusage.h`> but does not actually include <`sys/rusage.h`>.

Eight-bit Character Display in Non-internationalized XView Applications

XView applications that are not internationalized (that is, that do not use `XV_USE_LOCALE` or set it to `FALSE` in the `xv_init()` function) will continue to handle 8-bit characters. XView will run these applications under the `iso_8859_1` locale instead of the `C` locale, to maintain compatibility in 8-bit character handling. This change will impact the behavior of locale-sensitive functions such as the function `isprint`(3V), which now returns `TRUE` for 8-bit characters.

C Locale Display

The `C` locale does not support 8-bit characters (that is, G1 set or right-hand side of the ISO 8859-1 character set, such as characters with diareses). Use the `en_US` locale to display 8-bit characters in the English language environment.

D

Glossary

back-end input method

Refers to the architecture of an input method. With a back-end input method, a single X server connection is used. A dispatching mechanism must decide on this channel to delegate the appropriate keystrokes to the input method.

callback function

When an event occurs in an event-driven environment, a procedure is called. This procedure is called a callback function, or simply a callback. XView provides callback functions for handling and updating the input method status region and preedit region.

category, OPEN LOOK locale

The OPEN LOOK locale categories define the language and cultural conventions of an on-screen program. The categories consist of Basic Setting, Display Language, Input Language, Time Format, and Numeric Format.

character

A member of a set of elements used for the organization, control, or representation of text.

character set

A collection of characters. Character sets may be composed of alphabets, ideograms, or other elements.

CLE

Chinese Language Environment is an extension to SunOS 5.x and provides support for simplified Chinese.

coded character set

A character set whose characters are mapped to a bit representation. Some encodings, such as Compound Text, have more than one bit representation for a given character, and thus are not considered coded character sets.

codepoint

The coded representation of a single character in a coded character set.

code set

A list of unambiguous rules that establishes a character set and a one-to-one relationship between each character of the set and its bit representation. ASCII is the most common codeset; others examples are ISO 8859-1, JIS X0201, JIS X0208.

EUC-JIS

Used by EUC to mix ASCII, JIS X0201 and JIS X0208 character sets, which is popular in the Japanese UNIX market and adopted by JLE. In EUC-JIS, ASCII is defined in codeset 0, JIS X0208 is defined in codeset 1; and JIS X0201 (right half of the table only) is defined in codeset 2.

explicit commit

Refers to the process of sending or committing preedit text to the client application when the user presses the commit key or performs a specific commit key sequence.

Extended UNIX Code (EUC)

EUC is an encoding method that supports one primary code set and three supplementary code sets. The primary code set is always used to represent ASCII. The other three code sets vary depending upon the locale. EUC can be in either multibyte (EUC file code) or wide character (EUC process code) format. SunOS 5.x has adopted ATT's Multi-National Language Supplement (MNLS) EUC, which is patterned after ISO 2022.

file code (EUC file code)

A synonym for multibyte character. See *multibyte character.*

font set

A set of fonts representing the character sets of a language. In English, there is only one character set. Other languages, however, have multiple character sets that require multiple fonts. These multiple fonts are called font set objects.

HLE

Hanyu Language Environment is an extension to SunOS 5.x and provides support for traditional Chinese.

front-end input method

Refers to the architecture of an input method. With a front-end input method, there are two separate connections to the X server. Keystrokes go directly from the X server to the input method on one connection and other events go to the client connection. The input method acts as a filter and sends composed strings to the client. Synchronization is necessary between the two connections.

ICCCM

Inter-Client Communication Conventions Manual (ICCCM) describes conventions for the communication between X clients. This includes such conventions as client-to-client, client-to-window manager, client-to-session manager, and color characterization communication. This document is produced by the MIT X Consortium.

ideogram

A character representing an idea or thing without using a particular word of phrase for it.

implicit commit

Refers to the process of committing preedit text and sending the committed string to the client application, when the user performs certain common mouse or keyboard actions. This differs from explicit commit, in which a specific commit key sequence must be performed in order to commit preedit text.

input context (IC)

An abstraction for representing the state of a particular input thread for use with an input method. There may be multiple ICs associated with an input method.

input method (IM)

The algorithm by which users enter the text of a language. Input methods differ for each language depending on that language's structure and conventions.

input method server

A process that provides input method service to X clients. X input methods can be implemented either as a stub communicating to an input server or as a local library.

input method status

Input method (IM) status may consist of text data or bitmap data. The input method status is displayed in the input method status region and is updated when input method conversion is enabled or disabled, or when input modes change.

input style

Refers to the location of the preedit region during text input. In the *on-the-spot* input style, the preedit region is where the text will be inserted after it is committed. In *over-the-spot*, the preedit region is above where text will be inserted. *Root window* refers to input methods that use a preedit window that is a child of the Root Window.

internationalization

The capability of an application to be adapted to the requirements of different native languages, local customs and character sets.

JFP

Japanese Feature Package is an extension of SunOS 5.x. JFP uses the EUC encoding scheme and locale mechanism.

JLE

Japanese Language Environment.

KLE

Korean Language Environment is an extension to SunOS 5.x and provides support for Korean.

locale

A set of conventions unique to a geographical area and/or language.

locale setting

The process by which the user sets geographical area and/or language for the window system.

localization

The process of establishing information within a computer system that is specific to the operation of particular native languages, local customs and character sets.

lookup choice region

A screen region that displays alternate choices corresponding to the preedit string entered. The user selects the most appropriate lookup choice representation.

localized text handling

The method by which the native language strings of a program can be displayed in a foreign language without changing the program's source code.

multibyte character

A character whose codepoint is stored in one or more bytes. It differs from wide character encoding in that the number of bytes representing a character may vary.

object layout

The mechanism by which the position and dimension of objects containing strings may be modified to accommodate localized strings.

preediting

The process of composing characters from keystrokes. Preedit capability is a common feature of many input methods: the user types multiple keystrokes in order to compose a single character.

process code (EUC process code)

A synonym for wide character. See *wide character.*

wide character

A data type with fixed number of bytes in which a character from any supported character set is stored. Interpretation of a wide character is usually locale-dependent. ANSI-C uses a data type called `wchar_t` as the name of the data type.

Index

D

E

F

G

H

I

J

K

L

M

N

O

P

S

T

W

X